ORACLE

Gluster Storage for Oracle Linux: Best Practices and Sizing Guideline

Planning Gluster Storage for Oracle Linux

May, 2022

Public

Purpose statement

This document provides an overview of features and enhancements included in Gluster Storage for Oracle Linux. It is intended solely to help you assess the business and technical benefits of the product and better plan your IT projects.

Table of contents

Purpose statement 2
Introduction 4
Gluster terminology 4
Gluster Storage for Oracle Linux 5
Optimize Gluster Storage for Oracle Linux 8
Local disks, storage nodes, and network speed 8
Brick and XFS recommendations 10
Example - Creating multiple bricks on a physical device 12
Network 14
Memory 14
Virtual memory parameters 14
Small file performance enhancements 15
Best practices for tuning event threads 15
Setting the event threads value for a client 16
Setting the event thread value for a server 16
Verifying the event thread values 16
Other parameters related to even threads tuning 16
Enabling lookup optimization 16
Sizing and recommendation: Gluster Storage for Oracle Linux for video management system 17
Gluster Native Client, NFS, and NFS-Ganesha 18
Conclusion 19

Introduction

Gluster is an open source, scalable, distributed file system that aggregates disk storage resources from multiple servers into a single global namespace. There are several advantages to using Gluster such as:

- Scales to several petabytes
- Handles thousands of clients
- POSIX compatible
- Supports commodity hardware
- Can use any on-disk filesystem that supports extended attributes
- Accessible using industry standard protocols like NFS, iSCSI and SMB
- Provides replication, quotas, geo-replication, snapshots
- Allows optimization for different workloads

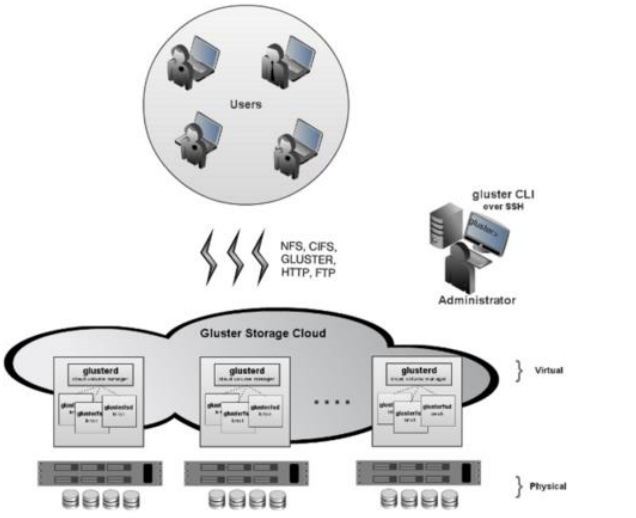

Image 1. Gluster Storage Example Architecture

Gluster is used in production at thousands of organizations spanning media, healthcare, government, education, web 2.0 and financial services sectors. Enterprises can use it to scale their capacity, performance, and availability on demand, without vendor lock-in, across on-premises, public cloud, and hybrid environments.

After years of inflexible proprietary hardware and appliance-based approaches, software-defined storage solutions have emerged as the viable alternative. The success of web-scale IT and public cloud providers has proven the way forward, with many embracing open software-defined storage as a fundamental strategy for deploying flexible, elastic, and cost-effective storage that is matched to specific application needs.

Gluster terminology

We recommend becoming familiar with the GlusterFS documentation, but to get started, the following terms are important to know:

- **Brick** (storage block): refers to the dedicated partition provided by the host for physical storage in the trusted host pool. It is the basic storage unit in GlusterFS, and also the storage directory provided by the server in the trusted storage pool.

- **Trusted storage pool**: A collection of shared files and directories based on the designed protocol.
- **Block storage**: Devices through which the data is being moved across systems in the form of blocks.
- **Cluster**: In Gluster storage, cluster and trusted storage pools convey the same meaning, the collaboration of storage servers based on a defined protocol.
- **Distributed file system**: A file system in which data is spread over different nodes, allowing users to access a file without knowing its location. Data sharing among multiple locations is fundamental to all distributed file systems.
- **Glusterd**: The GlusterFS management daemon and file system backbone that runs whenever the servers are in an active state.
- **POSIX**: Portable Operating System Interface (POSIX), the family of standards defined by the IEEE as a solution to the compatibility between Unix variants in the form of an API.
- **Volume**: A logical volume is a collection of bricks. A volume is a logical device for data storage, similar to the logical volume in Linux Logical Volume Manager (LVM). Most of the Gluster management operations are performed on the volume.
- **Subvolume**: A brick after being processed by least at one translator.
- **Translator**: A piece of code that performs the basic actions initiated by a user from the mount point. It connects one or more subvolumes.
- **Metadata**: Data providing information about other pieces of data.
- **Client**: The machine (or server) that mounts a volume.
- **FUSE** (file system in user space): a kernel module that allows users to create their own file systems without modifying kernel code.
- **VFS**: the interface provided by kernel space to user space to access disk.

Gluster Storage for Oracle Linux

Gluster Storage for Oracle Linux is a software-defined, open source solution, designed to meet unstructured and semi-structured data storage requirements, mainly focused on storing digital media files (images, video, and audio). Gluster Storage for Oracle Linux allows organizations to combine large numbers of storage and compute resources into a high-performance and centrally managed storage pool. The cluster can be scaled for increased capacity and/or increased performance. Gluster Storage for Oracle Linux capabilities include:

- **High availability and reliability**. When setting up GlusterFS, the type of storage volumes needs to be specified. The type chosen depends on the workload and the tradeoff between data protection and capacity. In typical deployments, both distributed and distributed-replicated volumes are used.
    - **Distributed volume**, the default type, spreads files across the bricks in the volume. It doesn't provide redundancy, but it's easy and inexpensive to scale the volume size. The downside is that a brick failure leads to complete data loss, and the underlying hardware must be relied on for data loss protection.

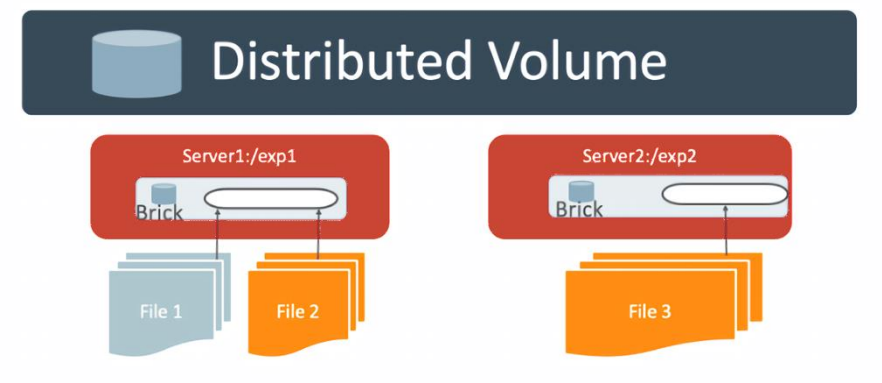

Image 2. Distributed volume setup with Gluster Storage

- **Replicated volume** creates copies of files across multiple bricks in the volume. Use replicated volumes in environments where high availability and high reliability are critical. A client-side quorum must be set on replicated volumes to prevent split-brain scenarios.

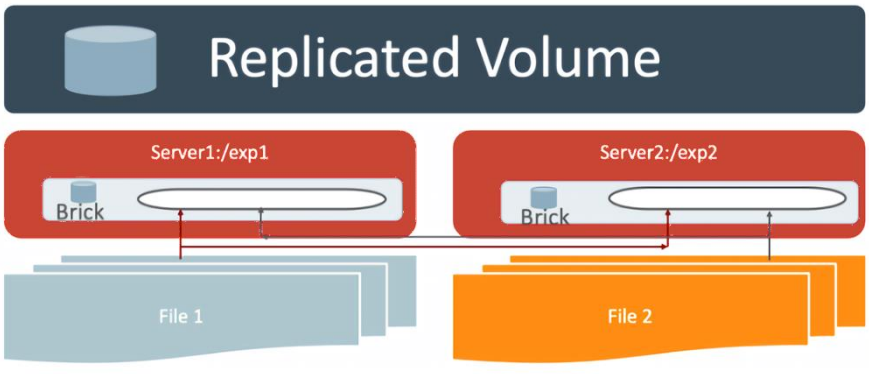

Image 3. Replicated volume setup with Gluster Storage

- **Distributed-replicated volume** provides node-level fault tolerance but less capacity than a distributed volume. Use distributed-replicated volumes in environments where the critical requirements are to scale storage and maintain high reliability. Distributed-replicated volumes also offer improved read performance in most environments.

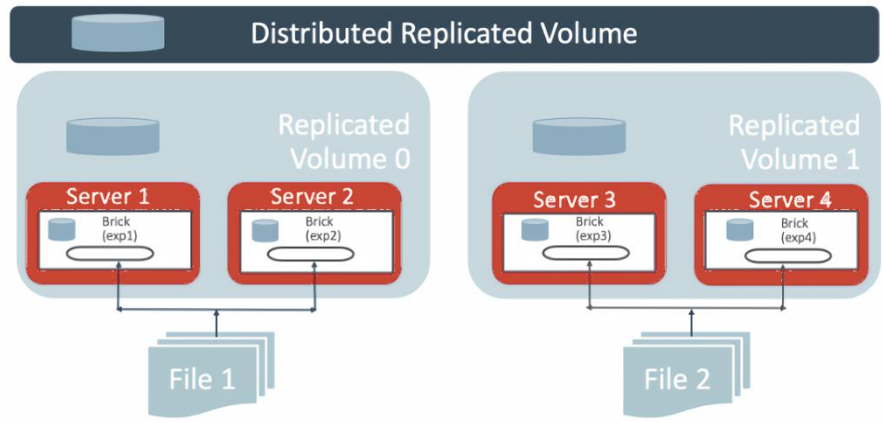

Image 4. Distributed-replicated volume setup with Gluster Storage

- **Dispersed volume** is based on erasure coding (EC). In this method of data protection, data is broken into fragments, expanded, and encoded with redundant data pieces, then stored across a set of different locations.

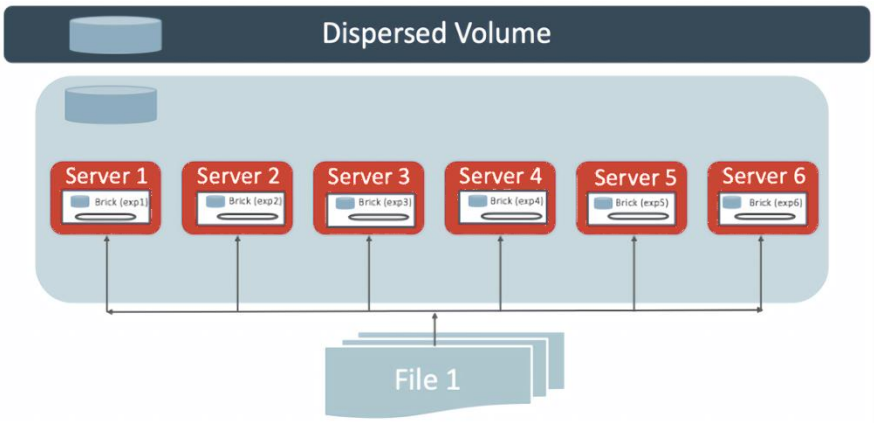

Image 5. Dispersed volume setup with Gluster Storage

- **Elastic volume management**. Gluster stores data in logical volumes that are independent of the logical storage pool. Logical storage pools can be added and removed online without causing business disruption. When volumes change, the data remains available without application interruption. With Gluster Storage for Oracle Linux, storage volumes are abstracted from the hardware and managed independently. The needs of the environment should be expanded or contracted elastically. In the multi-node scenario, the global uniform namespace can also do load balancing based on different nodes, which greatly improves access efficiency.
- **Persistent storage for containerized applications**. Containerized applications need persistent storage. In addition to bare metal, virtualized, and cloud environments, Gluster Storage for Oracle Linux can be deployed as persistent storage for Oracle Cloud Native Environment that includes a dedicated storage interface for Gluster.
- **Scalability**. Gluster Storage for Oracle Linux lets organizations start small and easily grow to support multi-petabyte repositories. *Gluster Elastic Hash* solves the dependence of Gluster on metadata server. Gluster uses Elastic Hash algorithm to locate data in storage pool. Gluster can intelligently locate any data fragment (store the data fragment on different nodes), without looking at the index or querying the metadata server. This design mechanism realizes the horizontal expansion of storage, improves the single point of failure and performance bottleneck, and realizes the parallel data access.
- **High performance**. Gluster Storage for Oracle Linux provides fast file access by eliminating the typical centralized metadata server. Files are spread throughout the storage nodes, helping to eliminate I/O bottlenecks and high latency. Organizations can use enterprise disk drives, local NVMe storage, and 10+ Gigabit Ethernet (GbE) to maximize distributed storage performance.
- **Industry-standard compatibility**. Gluster Storage for Oracle Linux service supports NFS, CIFS, HTTP, FTP, SMB, and Gluster native protocols, and is fully compatible with POSIX file systems standards. Existing applications can access the data in Gluster without any modification, or access it using a dedicated API, more efficient, which is very useful when deploying Gluster in a cloud environment.
- **Unified global namespace**. Gluster Storage for Oracle Linux aggregates disk and memory resources into a single common pool. This flexible approach simplifies management of the storage environment and helps

eliminate data silos. The global unified namespace aggregates all storage resources into a single virtual storage pool, which blocks the physical storage information for users and applications.

- **Cost-effective enterprise support**. Gluster Storage for Oracle Linux support is included in Oracle Linux Premier Support at no extra cost.

Optimize Gluster Storage for Oracle Linux

One of the benefits of Gluster is the ability to tailor storage infrastructure to different requirements and workload types. Local storage (rotating disks or NVMe storage) and network interface cards (NICs) are the hardware foundation where Gluster Storage for Oracle Linux resides. Based on hardware capacity, specific workloads and I/O requirements can be addressed. Multiple combinations are possible by varying:

- The number of Gluster Storage for Oracle Linux nodes
- The capacity and speed of local storage
- The capacity, in term of latency and bandwidth, of the network architecture (that includes also network switches)
- The layout of the underlying local storage, RAID configured or not
- The Gluster configuration, replicated or dispersed

The RAID configurations usually leveraged with Gluster Storage for Oracle Linux are RAID-6 and RAID-10:

- RAID-6 offers better disk space efficiency, good sequential write/read performance on large files; configured on 12 hardware disk devices RAID-6 can grant 40% more usable disk space while compared to RAID-10.
- RAID-10 offers improved performance for small-size files as well as random writes; RAID-10 is the preferred path for small-size files read and writes.

One important parameter, related to the hardware RAID configuration, is the stripe size.

With RAID 10, it's suggested to leverage a size of 256Kb striping while, for RAID-6, the striping size has to be defined by evaluating the number of hardware disk-devices leveraged. For RAID-6 configurations with 12 disks (10 dedicated to data), striping size suggested is 128Kb.

With JBOD (Just a Bunch of Disks) configuration the local disks are not aggregated by hardware RAID; JBOD supports 36 local-disks per storage node and, usually, one of those disks is leveraged as the cache for Gluster Storage.

"Replicated volumes" on RAID 6 bricks are usually leveraged for performance-optimized configurations, especially for workloads with a smaller file size while "dispersed volumes" on JBOD bricks are often more cost effective for large-file archive situations that do not have particular performance requirements. Among supported configurations, dispersed volumes can offer better read and write performance for workloads with large file sizes.

Standard 12 local-disk systems are often more performant and cost effective for smaller clusters and small-file applications, while dense 24 local-disk storage servers and larger are often more cost effective for larger clusters; Factors, including caching and tiering, with either standard SSDs or NVMe SSDs can provide significant benefits, especially for read performance. Also, leveraging full local storage based on SSD(s) or NVMe(s) can be evaluated, mostly while having high performance demand and low latency networking in place.

Local disks, storage nodes, and network speed

Local disks

In most scale-up systems, disk speed and seek time are the biggest determinants of performance. While the use of solid state drives or NVMe can eliminate the impact of spin time, such drives are generally more expensive per GB compared to low end SATA drives. In a typical Gluster configuration, the workload is spread across a large number of TB drives. Thus, the spin time of any individual drive becomes largely irrelevant. For example, Gluster has been able

to achieve upwards of 16 GB/s read throughput and 12 GB/s write throughput in an 8 nodes storage cluster created using 7.2 K RPM SATA drives. For most customers, therefore, the scale-out approach can eliminate the need for expensive drives or complicated tiering systems. In certain circumstances characterized by very high performance and low-capacity requirements (e.g. risk modeling), Gluster can be used in combination with SSD/NVMe for these workloads or in a full SSDs/NVMe solution for very demanding I/O throughput.

In most scale-out systems with a centralized or distributed metadata server, adding disks often leads to performance degradation or to non-linear gains in capacity. With Gluster, adding additional disks to a storage node will result in linear gains in effective capacity, and will generally result in either neutral to moderately positive impacts on performance.

Storage nodes

Gluster performance is most directly impacted by the number of storage nodes. In general, distributing the same number of disks among twice as many storage nodes will double performance. Performance in a Gluster cluster increases near-linearly with the number of storage nodes; an 8 storage nodes Gluster cluster will deliver approximately 4 times the throughput of a 2 storage nodes cluster.

Networking

In most cases, by the time a system has 8 or more storage nodes, the network becomes the bottleneck, and a 1 GbE system will be easily saturated. By adding a 10GbE or faster network, faster per node performance can be achieved. Gluster statistics show that it can achieve 16 GB/s read throughput and 12 GB/s write throughput in an 8 storage nodes cluster using low end SATA drives when configured with a 10GbE network. The same cluster achieved approximately 800 MB/s of throughput with a 1 GbE network.

Impact of disks, storage nodes, and networking components

To illustrate how Gluster Storage for Oracle Linux can scale, this example shows how a baseline system can be scaled to increase both performance and capacity:

- To support a requirement for 200 TB of capacity, a deployment might have 4 servers, each of which contains a quantity of 12 X 6 TB SATA drives.
- If performance levels are acceptable, but the desire is to increase the capacity by 33%, an option is to add another 4 X 6 TB drives to each server. This addresses the requirements and should not introduce performance degradation. (i.e., each server would have 16 X 6 TB drives). Note that an enterprise does not need to upgrade to larger, or more powerful hardware to increase capacity; it simply adds 8 more SATA drives.
- If the capacity is good, but the desire is to increase the performance by 100% (double-up) , an option is to distribute the drives among 8 servers, rather than 4, where each server would have 6 X 6 TB drives, rather than 12 X 6 TB,. Note that in this case, 2 more low-priced servers can be added, and existing drives would be redeployed.
- If the target is to both increase the performance by 100% (double-up) and also increase the capacity by 33%, as an example, then distributing among 8 servers where each server has 16 X 6 TB drives will address the requirement.

As shared above, the power of Gluster Storage for Oracle Linux scalability covers both the capacity and the performance where both can scale linearly to meet requirements. It is not necessary to know what performance levels will be needed in the future, since the Gluster Storage for Oracle Linux configuration can be easily adapted based on new requirements.

Brick and XFS recommendations

Format bricks using the following configurations to enhance performance:

The steps for creating a brick and multiple bricks from a physical device are listed below.

- **Creating the physical volume**

The `pvcreate` command is used to create the physical volume. The Logical Volume Manager can use a portion of the physical volume for storing its metadata while the rest is used as the data portion. Align the I/O at the Logical Volume Manager (LVM) layer using `--dataalignment` option while creating the physical volume.

The command is used in the following format:

```
# pvcreate --dataalignment alignment_value disk
```

For JBOD, use an alignment value of `256K`.

In case of hardware RAID, the `alignment_value` should be obtained by multiplying the RAID stripe unit size with the number of data disks. If 12 disks are used in a RAID 6 configuration, the number of data disks is 10; on the other hand, if 12 disks are used in a RAID 10 configuration, the number of data disks is 6.

For example, the following command is appropriate for 12 disks in a RAID 6 configuration with a stripe unit size of 128 KiB:

```
# pvcreate --dataalignment 1280k disk
```

The following command is appropriate for 12 disks in a RAID 10 configuration with a stripe unit size of 256 KiB:

```
# pvcreate --dataalignment 1536k disk
```

To view the previously configured physical volume settings for `--dataalignment`, run the following command:

```
# pvs -o +pe_start disk
  PV         VG   Fmt  Attr PSize PFree 1st PE
  /dev/sdb        lvm2 a--  9.09t 9.09t   1.25m
```

- **Creating the volume group**

The volume group is created using the `vgcreate` command.

For hardware RAID, in order to help ensure that logical volumes created in the volume group are aligned with the underlying RAID geometry, it is important to use the `-- physicalextentsize` option. Execute the `vgcreatecommand` in the following format:

```
# vgcreate --physicalextentsize extent_size VOLGROUP physical_volume
```

The `extent_size` should be obtained by multiplying the RAID stripe unit size with the number of data disks. If 12 disks are used in a RAID 6 configuration, the number of data disks is 10; on the other hand, if 12 disks are used in a RAID 10 configuration, the number of data disks is 6.

For example, run the following command for RAID-6 storage with a stripe unit size of 128 KB, and 12 disks (10 data disks):

```
# vgcreate --physicalextentsize 1280k VOLGROUP physical_volume
```

In the case of JBOD, use the `vgcreate` command in the following format:

```
# vgcreate VOLGROUP physical_volume
```

- **Creating the thin pool**

A thin pool provides a common pool of storage for thin logical volumes (LVs) and their snapshot volumes, if any.

Execute the following command to create a thin-pool:

```
# lvcreate --thinpool VOLGROUP/thin_pool --size <pool_size> --chunksize <chunk_size> --
poolmetadatasize <meta_size> --zero n
```

- `poolmetadatasize`

Internally, a thin pool contains a separate metadata device that is used to track the (dynamically) allocated regions of the thin LVs and snapshots. The `poolmetadatasize` option in the above command refers to the size of the pool meta data device.

The maximum possible size for a metadata LV is 16 GiB. The recommendation for GlusterFS is to create the metadata device of the maximum supported size. If space is a concern, less than the maximum size can be allocated, but in this case a minimum of 0.5% of the pool size should be allocated.

- `chunksize`

An important parameter to be specified while creating a thin pool is the chunk size, which is the unit of allocation. For good performance, the chunk size for the thin pool and the parameters of the underlying hardware RAID storage should be chosen so that they work well together.

For RAID 6 storage, the striping parameters should be chosen so that the full stripe size (stripe_unit size * number of data disks) is between 1 MiB and 2 MiB, preferably in the low end of the range. The thin pool chunk size should be chosen to match the RAID 6 full stripe size. Matching the chunk size to the full stripe size aligns thin pool allocations with RAID 6 stripes, which can lead to better performance. Limiting the chunk size to below 2 MiB helps reduce performance problems due to excessive copy-on-write when snapshots are used.

For example, for RAID 6 with 12 disks (10 data disks), stripe unit size should be chosen as 128 KiB. This leads to a full stripe size of 1280 KiB (1.25 MiB). The thin pool should then be created with the chunk size of 1280 KiB.

For RAID 10 storage, the preferred stripe unit size is 256 KiB. This can also serve as the thin pool chunk size. Note that RAID 10 is recommended when the workload has a large proportion of small file writes or random writes. In this case, a small thin pool chunk size is more appropriate, as it reduces copy-on-write overhead with snapshots.

For JBOD, use a thin pool chunk size of 256 KiB.

- `block zeroing`

By default, the newly provisioned chunks in a thin pool are zeroed to prevent data leaking between different block devices. In the case of Gluster, where data is accessed via a file system, this option can be turned off for better performance with the `--zero n` option. Note that `n` does not need to be replaced. The following example shows how to create the thin pool:

```
# lvcreate --thinpool VOLGROUP/thin_pool --size 800g --chunksize 1280k --poolmetadatasize 16G
--zero n
```

- **Creating a thin logical volume**

After the thin pool has been created as mentioned above, a thinly provisioned logical volume can be created in the thin pool to serve as storage for a brick of a Gluster volume.

```
# lvcreate --thin --name LV_name --virtualsize LV_size VOLGROUP/thin_pool
```

Example - Creating multiple bricks on a physical device

The steps above (LVM Layer) cover the case where a single brick is being created on a physical device.

Note: In these steps, we are assuming the following:

- *Two bricks must be created on the same physical device*
- *One brick must be of size 4 TiB and the other is 2 TiB*
- *The device is /dev/sdb, and is a RAID-6 device with 12 disks*
- *The 12-disk RAID 6 device has been created according to the recommendations in this chapter, that is, with a stripe unit size of 128 KiB*

This example shows how to adapt these steps when multiple bricks need to be created on a physical device.

- Create a single physical volume using pvcreate

```
# pvcreate --dataalignment 1280k /dev/sdb
```

- Create a single volume group on the device

```
# vgcreate --physicalextentsize 1280k vg1 /dev/sdb
```

- Create a separate thin pool for each brick using the following commands

```
# lvcreate --thinpool vg1/thin_pool_1 --size 4T --chunksize 1280K --poolmetadatasize 16G --
zero n
# lvcreate --thinpool vg1/thin_pool_2 --size 2T --chunksize 1280K --poolmetadatasize 16G --
zero n
```

In the examples above, the size of each thin pool is chosen to be the same as the size of the brick that will be created in it. With thin provisioning, there are many possible ways of managing space.

- Create a thin logical volume for each brick

```
# lvcreate --thin --name lv1 --virtualsize 4T vg1/thin_pool_1
# lvcreate --thin --name lv2 --virtualsize 2T vg1/thin_pool_2
```

- Follow the XFS recommendations (next step) in this sectionfor creating and mounting file systems for each of the thin logical volumes

```
# mkfs.xfs options /dev/vg1/lv1
# mkfs.xfs options /dev/vg1/lv2
# mount options /dev/vg1/lv1 mount_point_1
# mount options /dev/vg1/lv2 mount_point_2
```

- XFS inode Size

As GlusterFS makes extensive use of extended attributes, an XFS inode size of 512 bytes works better with GlusterFS than the default XFS inode size of 256 bytes. So, inode size for XFS must be set to 512 bytes while formatting the GlusterFS bricks. To set the inode size, use -i size option with the `mkfs.xfs` command as shown in the following Logical Block Size for the Directory section.

- XFS RAID Alignment

When creating an XFS file system, the striping parameters of the underlying storage can be explicitly specified in the following format:

```
# mkfs.xfs other_options -d su=stripe_unit_size,sw=stripe_width_in_number_of_disks device
```

For RAID 6, ensure that I/O is aligned at the file system layer by providing the striping parameters. For RAID 6 storage with 12 disks, if the recommendations above have been followed, the values must be:

```
# mkfs.xfs other_options -d su=128k,sw=10 device
```

For RAID 10 and JBOD, the `-d su=<>,sw=<>` option can be omitted. By default, XFS will use the thin-p chunk size and other parameters to make layout decisions.

- Logical block size for the directory

An XFS file system allows a logical block size to be selected for the file system directory that is greater than the logical block size of the file system. Increasing the logical block size for the directories from the default 4 K, decreases the directory I/O, which in turn improves the performance of directory operations. To set the block size, use `-n size` option with the `mkfs.xfs` command.

Following is the example output of RAID 6 configuration along with inode and block size options:

```
# mkfs.xfs -f -i size=512 -n size=8192 -d su=128k,sw=10 logical volume
meta-data=/dev/mapper/gluster-brick1 isize=512    agcount=32, agsize=37748736 blks
         =    sectsz=512   attr=2, projid32bit=0
data     =     bsize=4096   blocks=1207959552, imaxpct=5
         =    sunit=32     swidth=320 blks
naming   = version 2   bsize=8192   ascii-ci=0
log      =internal log   bsize=4096   blocks=521728, version=2
         =    sectsz=512   sunit=32 blks, lazy-count=1
realtime =none    extsz=4096   blocks=0, rtextents=0
```

- Allocation strategy

`inode32` and `inode64` are the two most common allocation strategies for XFS. With inode32 allocation strategy, XFS places all the `inodes` in the first 1 TiB of disk. With a larger disk, all the `inodes` would be stuck in the first 1 TiB. `inode32` allocation strategy is used by default.

With `inode64` mount option `inodes` would be replaced near to the data which would minimize the disk seeks.

To set the allocation strategy to `inode64` when the file system is being mounted, use the `-o inode64 option with the mount` command as shown in the following `Access Time` section.

- Access time

If the application does not require an update to the access time on files, then the file system must be mounted with `noatime` mount option. For example:

```
# mount -t xfs -o inode64,noatime <logical volume> <mount point>
```

This optimization improves performance of small-file reads by avoiding updates to the XFS inodes when files are read.

```
/etc/fstab entry for option E + F
 <logical volume> <mount point>xfs     inode64,noatime   0 0
```

- `Allocation groups`

Each XFS file system is partitioned into regions called allocation groups. Allocation groups are similar to the block groups in ext3, but allocation groups are much larger than block groups and are used for scalability and parallelism rather than disk locality. The default allocation for an allocation group is 1 TiB.

Allocation group count must be large enough to sustain the concurrent allocation workload. In most of the cases allocation group count chosen by `mkfs.xfs` command would provide optimal performance. Do not change the allocation group count chosen by `mkfs.xfs`, while formatting the file system.

- `Percentage of space allocation to inodes`

If the workload is very small files (average file size is less than 10 KB), then it is recommended to set `maxpct` value to `10`, while formatting the file system.

For small-file and random write performance, using writeback cache is strongly recommend, that is, non-volatile random-access memory (NVRAM) in the storage controller. For example, typical Dell and HP storage controllers have it. Ensure that NVRAM is enabled, that is, the battery is working. Refer to the hardware documentation for details on enabling NVRAM.

Do not enable writeback caching in the disk drives, this is a policy where the disk drive considers the write is complete before the write actually made it to the magnetic media (platter). As a result, the disk write cache might lose its data during a power failure leading to file system corruption.

Network

Data traffic on the network becomes a bottleneck as and when the number of storage nodes increase. By adding a 10GbE or faster network for data traffic, faster per node performance can be achieved. Jumbo frames must be enabled at all levels, that is, client, Gluster node, and ethernet switch levels. `MTU` of size `N+208` must be supported by the ethernet switch where `N=9000`. It is recommended that there is a separate network for management and data traffic when protocols like NFS /CIFS are used instead of native client. Preferred bonding mode for Gluster client is mode 6 (`balance-alb`), this allows the client to transmit writes in parallel on separate NICs much of the time.

Memory

Gluster does not consume significant compute resources from the storage nodes themselves. However, read-intensive workloads can benefit greatly from additional RAM.

Virtual memory parameters

The data written by the applications is aggregated in the operating system page cache before being flushed to the disk. The aggregation and writeback of dirty data is governed by the Virtual Memory parameters. The following parameters may have a significant performance impact:

- `vm.dirty_ratio`
- `vm.dirty_background_ratio`

The appropriate values of these parameters vary with the type of workload:

- Large-file sequential I/O workloads benefit from higher values for these parameters.
- For small-file and random I/O workloads it is recommended to keep these parameter values low.

The Gluster tuned profiles set the values for these parameters appropriately. Hence, it is important to select and activate the appropriate Gluster profile based on the workload.

Small file performance enhancements

For each file access, an operation via the network is executed on Gluster metadata for files and directories; this process is called "*lookup*". Due to this in-place process, a heavy workload on many small files performs poorly compared to other types of workloads. With big files, most of the time is dedicated to the file transfer while for small files, the metadata management could take more time than the storage operation.

Metadata intensive workloads (small files, high concurrency) require some optimization on both the network and the storage to minimize the possibility of slow throughput and response time;

To address this kind of limitation for small files, one option is to enhance the Gluster metadata cache translator (**md-cache**), so the *lookup* requests are cached indefinitely on the client. This solution needs the client-side cache to be invalidated (*upcall*) if files or directories are modified in the meantime by other clients. With upcalls enabled, the number of lookups drops to a lower number on all subvolumes. This drop grants an improved throughput for small file workloads.

To enable upcall in **md-cache**, specify the following:

```
# gluster volume set <volname> features.cache-invalidation on
# gluster volume set <volname> features.cache-invalidation-timeout 600
# gluster volume set <volname> performance.cache-samba-metadata on
# gluster volume set <volname> performance.cache-invalidation on
# gluster volume set <volname> performance.stat-prefetch on
```

The following parameters can help fine tune file access for small-file sizes. These examples are for reference and may need further tuning to obtain the best performance on the specified environment:

```
# gluster volume set <volname> client.event-threads 4
# gluster volume set <volname> server.event-threads 4
# gluster volume set <volname> performance.io-thread-count  64
# gluster volume set <volname> server.outstanding-rpc-limit: 128
# gluster volume set <volname> cluster.lookup-optimize ontuned-adm profile throughput-
performance
```

Best practices for tuning event threads

Performance improvements for Gluster Storage for Oracle Linux could be obtained by tuning the number of threads processing events from network connections. Here some tuning recommendations related to event thread values.

- One thread manages one single connection at a time; having more threads than connections, either on Gluster server(s) or the Gluster client is not recommended.
- Introducing, on both Gluster server and client, a number of threads higher than the number of CPU core(s) available on the system could cause context switches with the result of slower performance.
- In the case of a single thread consuming high percentages of CPU time, the increase of the even thread value could improve the performance and response time of the Gluster server.

Setting the event threads value for a client

GlusterFS Server performance can be tuned using the event thread values.

```
# gluster volume set VOLNAME client.event-threads <value>
# gluster volume set test-vol client.event-threads 4
```

Setting the event thread value for a server

GlusterFS Server performance can be tuned using event thread values.

```
# gluster volume set VOLNAME server.event-threads <value>
# gluster volume set test-vol server.event-threads 4
```

Verifying the event thread values

Verify the event thread values that are set for the client and server components by executing the following command:

```
# gluster volume info VOLNAME
```

Other parameters related to even threads tuning

The following parameters may also help on Gluster Storage for Oracle Linux:

- `server.outstanding-rpc-limit` configuration which queue the requests for brick processes
- `performance.io-thread-count` configuration which performs the actual IO operations

For further details related to these two parameters, please refer to GlusterFS community documentation.

Enabling lookup optimization

A lookup for a file/directory that does not exist is a negative lookup; negative lookups are expensive and typically slow down file creation, as DHT(Distributed Hash Table) attempts to find the file in all subvolumes. This especially impacts small file performance, where a large number of files are being added/created in quick succession to the volume.

The negative lookup fan-out behavior can be optimized by not performing the lookup process in a balanced volume.

The `cluster.lookup-optimize` configuration option enables lookup optimization. To enable this option run the following command:

```
# gluster volume set VOLNAME cluster.lookup-optimize <on/off>
```

Note: *The configuration takes effect for newly created directories immediately after setting the above option. For existing directories, a rebalance is required to help ensure the volume is in balance before DHT applies the optimization on older directories.*

Sizing and recommendation: Gluster Storage for Oracle Linux for video management system

This paper provides a representation of use cases and respective best practices for Gluster Storage for Oracle Linux volumes. Gluster volumes can be used productively for a large variety of data workloads, starting from development storage solution requirements to production and intensive I/O requirements.

A video management system, such as streaming HD video recording and management, usually involves a sequential write-heavy workload that is sensitive to overall latency. It also includes accompanying concurrent read workloads that may be smaller and random in nature; usually writes cover 80% of the workload while reads only cover the remaining 20%. The in-place architecture has to be able to grant the required number of simultaneous high-definition recording streams (writes) before reaching any data loss.

Based on the best practices shared in earlier sections, it's evident that the replicated volume on RAID 6 bricks is not the best option for this kind of use case. A dispersed volume on JBOD bricks provides the advantages of additional storage and higher capacity, in term of the number of write streams clients.

To avoid data loss in case of local disk failure (JBOD does not preserve on this), the "dispersed volume" type is suggested. This option leverages local NVMe storage as well as high bandwidth networking (10+G/bit devices).

At the system level, tuned profile `throughput-performance` is suggested; `throughput-performance` disables power saving mechanisms and enables sysctl settings that improve the throughput performance of disk and network IO. CPU governor is set to performance and CPU energy performance bias is set to performance. Disk readahead values are increased.

Other recommendations are also related to Gluster volume as well as Gluster clients.

For Gluster Storage for Oracle Linux volume:

- `disperse.eager-lock`
- Suggested value: `off`

If eager-lock is on, the lock remains in place either until lock contention is detected, or for 1 second in order to check if there is another request for that file from the same client. If eager-lock is off, locks release immediately after file operations complete, improving performance for some operations, but reducing access efficiency.

- `cluster.lookup-optimize`
- Suggested value: `on`

This option enables the optimization of -ve lookups, by not doing a lookup on non-hashed subvolumes for files, in case the hashed subvolume does not return any result. This option disregards the `lookup-unhashed` setting, when enabled. `lookup-unhashed` does a lookup through all the subvolumes, in case a lookup didn't return any result from the hashed subvolume. If set to OFF, it does not do a lookup on the remaining subvolumes.

- `performance.client-io-threads`
- Suggested value: `on`

Performance improves for parallel I/O from a single mount point for dispersed (erasure-coded) volumes by allowing up to 16 threads to be used in parallel. When enabled, 1 thread is used by default, and further threads up to the maximum of 16 are created as required by the client workload.

- `server.event-threads`
- Suggested value: `4 or more (32 is the max limit)`

Specifies the number of network connections to be handled simultaneously by the server processes.

- client.event-threads
- Suggested value: `4 ore more (32 is the max limit)`

Specifies the number of network connections to be handled simultaneously by the client processes accessing Gluster Storage for Oracle Linux.

For Oracle Linux Gluster clients:

- `net.core.netdev_max_backlog`
- Suggested value: `131072 or higher`

Increases number of incoming connections backlog queue. Sets the maximum number of packets, queued on the INPUT side, when the interface receives packets faster than the kernel can process them.

- `sunrpc.tcp_slot_table_entries`
- Suggested value with 10 GigE network: `48 or higher` depending on the round-trip time

`sunrpc.tcp_slot_table_entries` sets the number of (TCP) minimum RPC entries to pre-allocate for in-flight RPC requests

- `net.core.somaxconn`
- Suggested value: `check number of cores available and requirements`

Each CPU core can hold a number of packets in a ring buffer before the network stack is able to process them. If the buffer is filled faster than the TCP stack can process them, a dropped packet counter is incremented and they will be dropped. The `net.core.netdev_max_backlog` setting should be increased to maximize the number of packets queued for processing on clients with high burst traffic. It's important to remember that `net.core.netdev_max_backlog` is a per CPU core setting. The default value for `net.core.somaxconn` comes from the SOMAXCONN constant, which is set to 128 on the Unbreakable Enterprise Kernel (UEK) 5 for Oracle Linux (or upstream kernels lower than 5.4), while SOMAXCONN had been raised to 4096 in UEK6.

- `net.ipv4.tcp_fin_timeout`
- Suggested value: `5`

The length of time an orphaned (no longer referenced by any application) connection will remain in the FIN_WAIT_2 state before it is aborted at the local end.

Gluster Native Client, NFS, and NFS-Ganesha

Gluster's Native FUSE client will deliver better read and write performance than Gluster NFS across a wide variety of block sizes. However, Gluster NFS will deliver better write performance at small (<32 K) block sizes because of its kernel-based write caching. Native clients write performance is sensitive to changes in block size, and is not sensitive to changes in file size. For NFS, both read and write performance are not sensitive to changes in block size, but are sensitive to changes in file size. The reason that this happens is that NFS clients perform write caching in addition to read caching, while the Gluster native client only caches the reads.

Generic performance benchmarks demonstrated a discrete performance improvement while leveraging NFS-Ganesha over the standard Kernel NFS Server. Introducing the adoption of NSF-Ganesha server-side (on Gluster Storage for Oracle Linux nodes) does not introduce any impact or requirement on NFS clients.

Conclusion

As these Gluster Storage for Oracle Linux guidelines and considerations show, a little up-front work to plan the environment is well worth the time investment as it can help ensure service level commitments are met and services can be delivered on time and on budget.

Gluster Storage is available on the Unbreakable Linux Network (ULN) and the Oracle Linux yum server. For more information on hardware requirements and how to install and configure Gluster, please review the Gluster Storage for Oracle Linux Documentation.

For more information about Oracle Linux, visit oracle.com/linux and blogs.oracle.com/linux.

Connect with us

Call +**1.800.ORACLE1** or visit **oracle.com/linux**. Outside North America, find your local office at: **oracle.com/contact**.

blogs.oracle.com facebook.com/oracle twitter.com/oracle