



Cloud Storage for the Modern Data Center

An Introduction to
Gluster Architecture

Versions 3.1.x

TABLE OF CONTENTS

Table of Contents.....	2
Abstract.....	3
1.0 Gluster Design Goals.....	4
1.1 Elasticity	4
1.2 Linear Scaling.....	4
1.3 Scale-out	5
2.0 Technical Differentiators.....	9
2.1 Software Only	9
2.2 Open Source	9
2.3 Complete Storage Operating System Stack.....	9
2.4 User Space	9
2.5 Modular, Stackable Architecture	9
2.6 Data Stored in Native Formats	10
2.7 No Metadata with the Elastic Hash Algorithm	10
3.0 The Elastic Hashing Algorithm	12
3.1 Centralized Metadata Systems.....	12
3.2 Distributed Metadata Systems.....	13
3.3 An Algorithmic Approach (no metadata model)	14
3.4 The Use of Hashing.....	16
3.5 Making it all Elastic.....	17
3.6 Advanced Topics	18
Elastic Volume Management	18
Renaming or Moving Files	18
High Availability.....	18
4.0 Conclusion.....	19
5.0 Glossary.....	20
6.0 References and Further Reading.....	21

ABSTRACT

Over the past ten years, enterprises have seen enormous gains as they migrated from proprietary, monolithic server architectures to architectures that are virtualized, open source, standardized, and commoditized.

Unfortunately, storage has not kept pace with computing. The proprietary, monolithic, and scale-up solutions that dominate the storage industry today do not deliver the economics, flexibility, and increased scaling capability that the modern data center needs in a hyper growth, virtualized, and cloud-based world. Gluster was created to address this gap.

Gluster delivers scale-out NAS for virtual and cloud environments.

Gluster is a file-based scale-out NAS platform that is open source and software only. It allows enterprises to combine large numbers of commodity storage and compute resources into a high performance, virtualized and centrally managed pool. Both capacity and performance can scale independently on demand, from a few terabytes to multiple petabytes, using both on-premise commodity hardware and public cloud storage infrastructure. By combining commodity economics with a scale-out approach, customers can achieve radically better price and performance, in an easy-to-manage solution that can be configured for the most demanding workloads.

This document discusses some of the unique technical aspects of the Gluster architecture, discussing those aspects of the system that are designed to provide linear scale-out of both performance and capacity without sacrificing resiliency. Particular attention is paid to the Gluster Elastic Hashing Algorithm.

1.0 GLUSTER DESIGN GOALS

Gluster was designed to achieve several major goals:

1.1 ELASTICITY

Elasticity is the notion that an enterprise should be able to flexibly adapt to the growth (or reduction) of data and add or remove resources to a storage pool as needed, without disrupting the system. Gluster was designed to allow enterprises to add or delete volumes & users, and to flexibly add or delete virtual machine (VM) images, application data, etc., without disrupting any running functionality.

1.2 LINEAR SCALING

Linear Scaling is a much-abused phrase within the storage industry. It should mean, for example, that twice the amount of storage systems will deliver twice the observed performance: twice the throughput (as measured in gigabytes per second), with the same average response time per external file system I/O event (i.e., how long will an NFS client wait for the file server to return the information associated with each NFS client request).

Similarly, if an organization has acceptable levels of performance, but wants to increase capacity, they should be able to do so without decreasing performance or getting non-linear returns in capacity.

Unfortunately, most storage systems do not demonstrate linear scaling. This seems somewhat counter-intuitive, since it is so easy to simply purchase another set of disks to double the size of available storage. The caveat in doing so is that the scalability of storage has multiple dimensions, capacity being only one of them.

Adding capacity is only one dimension; the systems managing those disks need to scale as well. There needs to be enough CPU capacity to drive all of the spindles at their peak capacity, the file system must scale to support the total size, the metadata telling the system where all the files are located must scale at the same rate disks are added, and the network capacity available must scale to meet the increased number of clients accessing those disks. In short, it is not storage that needs to scale as much as it is the complete storage system that needs to scale.

Traditional file system models and architectures are unable to scale in this manner and therefore can never achieve true linear scaling of performance. For traditional distributed systems, each node must always incur the overhead of interacting with one or more other nodes for every file operation, and that overhead subtracts from the scalability simply by adding to the list of tasks and the amount of work to be done.

Even if those additional tasks could be done with near-zero effort (in the CPU and other system-resource sense of the term), latency problems remain. Latency results from waiting for the responses across the networks connecting the distributed nodes in those traditional system architectures and nearly always impacts performance. This type of latency increases proportionally relative to the speed and responsiveness - or lack of - of the networking connecting the nodes to each other. Attempts to minimize coordination overhead often result in unacceptable increases in risk.

This is why claims of linear scalability often break down for traditional distributed architectures.

Instead, as illustrated in Figure1, most traditional systems demonstrate logarithmic scalability--storage's useful capacity grows more slowly as it gets larger. This is due to the increased overhead necessary to maintain data resiliency. Examining the performance of some storage networks reflects this limitation as larger units offer slower aggregate performance than their smaller counterparts.

1.3 SCALE-OUT WITH GLUSTER

Gluster is designed to provide a scale-out architecture for both performance and capacity. This implies that the system should be able to scale up (or down) along multiple dimensions.



Figure 1 Linear vs. Logarithmic Scaling

By aggregating the disk, CPU, and I/O resources of large numbers of inexpensive systems—an enterprise should be able to create one very large and performant storage pool. If the enterprise wants to add more capacity to a scale-out a system, they can do so by adding more inexpensive disks. If the enterprise wants to gain performance, they can do so by deploying disks between more inexpensive sever nodes.

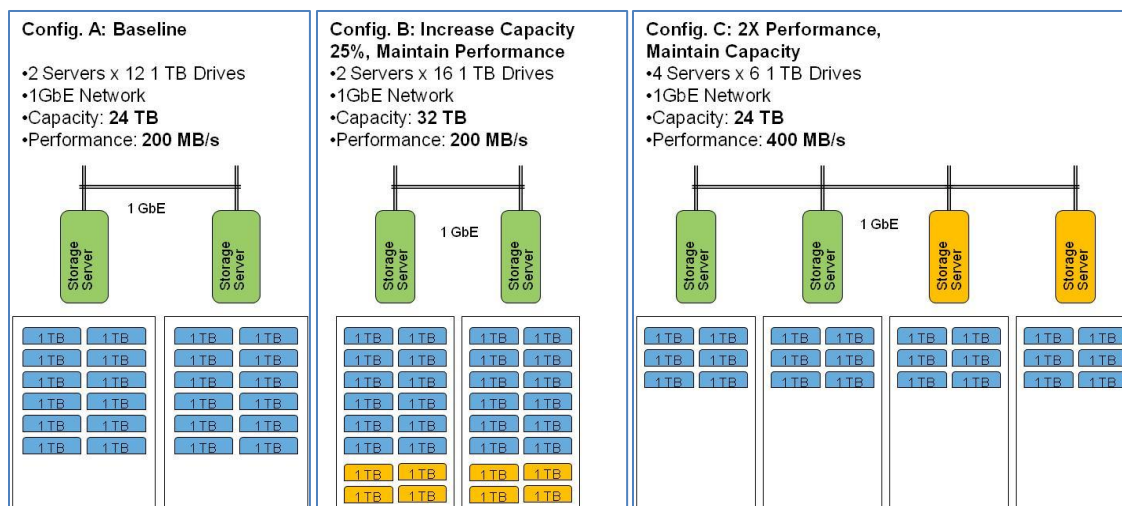
Gluster's unique architecture is designed to deliver the benefits of scale-out (more units = more capacity, more CPU, and more I/O), while avoiding the corresponding overhead and risk associated with keeping large numbers of nodes in synch.

In practice, both performance and capacity can be scaled out linearly in Gluster. We do this by employing three fundamental techniques:

1. The elimination of metadata
2. Effective distribution of data to achieve scalability and reliability.
3. The use of parallelism to maximize performance via a fully distributed architecture

To illustrate how Gluster scales, Figure 2, below shows how a baseline system can be scaled to increase both performance and capacity. The discussion below uses some illustrative performance and capacity numbers. For a more complete discussion of performance scaling with Gluster, with detailed results from actual tests, please see the document, "Scaling Performance in a Gluster Environment."

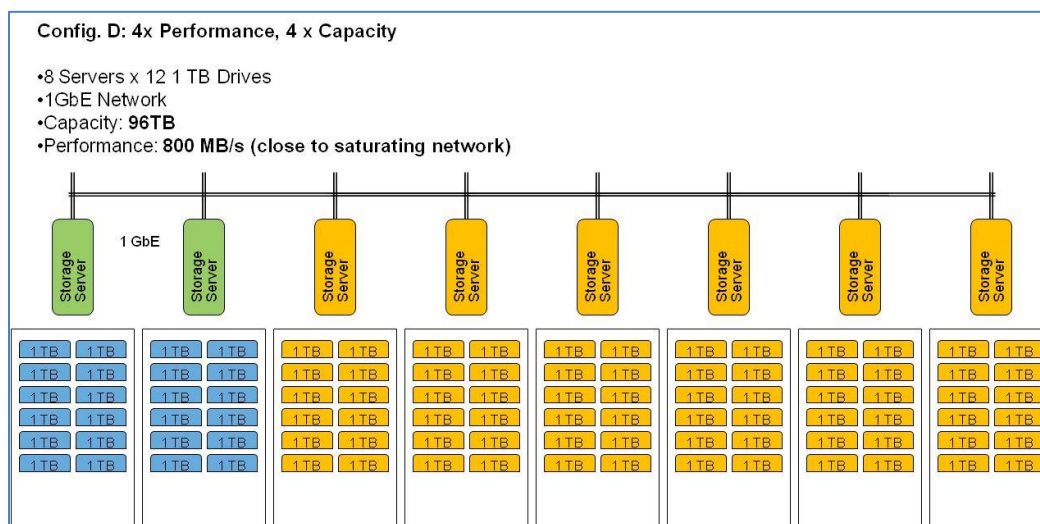
A typical direct attached Gluster configuration will have a moderate number of disks attached to 2 or more server nodes which act as NAS heads. For example, to support a requirement for 24 TB of capacity, a deployment might have 2 servers, each of which contains a quantity of 12, 1 TB SATA drives. (See Config A, below).



If a customer has found that the performance levels are acceptable, but wants to increase capacity by 25%, they could add another 4, 1 TB drives to each server, and will not generally experience performance degradation. (i.e., each server would have 16, 1 TB drives). (See Config. B, above). Note that they do not need to upgrade to larger, or more powerful hardware, they simply add 8 more inexpensive SATA drives.

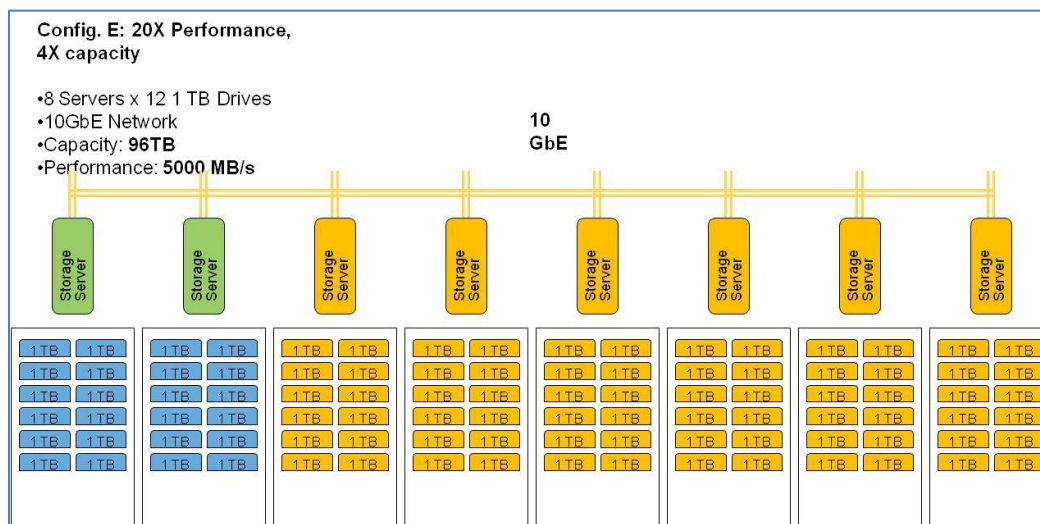
On the other hand, if the customer is happy with 24 TB of capacity, but wants to double performance, they could distribute the drives among 4 servers, rather than 2 servers (i.e. each server would have 6, 1 TB drives, rather than 12). Note that in this case, they are adding 2 more low-price servers, and can simply redeploy existing drives. (See Config. C, above)

If they want to both quadruple performance and quadruple capacity, they could distribute among 8 servers (i.e. each server would have 12, 1 TB drives). (See Config. D, below)



Note that by the time a solution has approximately 10 drives, the performance bottleneck has generally already moved to the network. (See Config. D, above)

So, in order to maximize performance, we can upgrade from a 1 Gigabit Ethernet network to a 10 Gigabit Ethernet network. Note that performance in this example is more than 25x that which we saw in the baseline. This is evidenced by an increase in performance from 200 MB/s in the baseline configuration to 5,000 MB/s. (See Config. E, below)



As you will note, the power of the scale-out model is that both capacity and performance can scale linearly to meet requirements. It is not necessary to know what performance levels will be needed 2, or 3 years out. Instead, configurations can be easily adjusted as the need demands.

While the above discussion was using round, theoretical numbers, actual performance tests have proven out this linear scaling. The results below in Figure 2 show write throughput scaling linearly from 100 MB/s on one server (e.g. storage system) to 800 MB/s (on 8 systems) in a 1 GbE environment. **However, on an Infiniband network, we have seen write throughput scale from 1.5 GB/s (one system) to 12 GB/s (8 systems).**

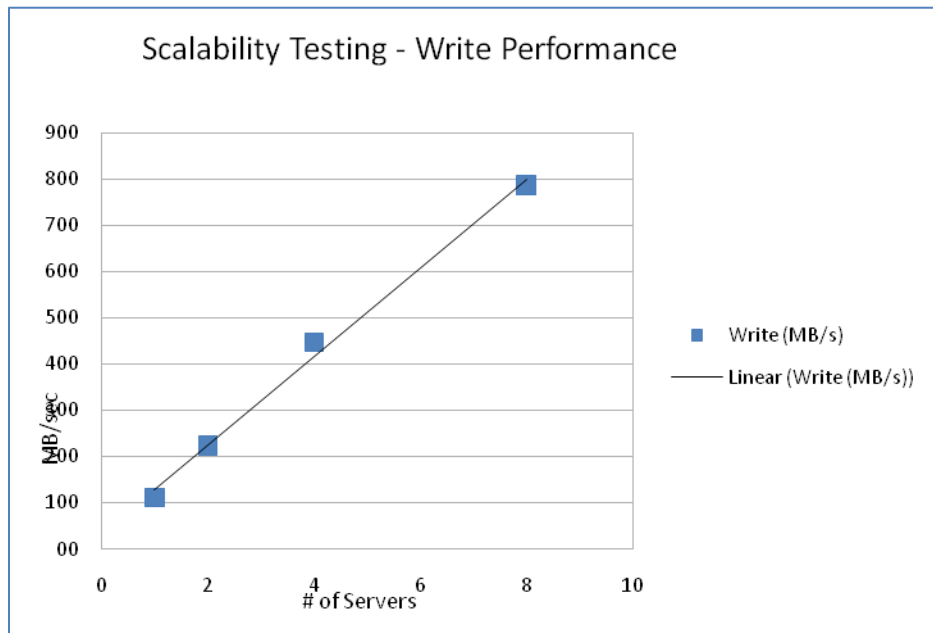


Figure 2: Linear Scaling In Gluster¹

We have experience with Gluster being deployed in a multitude of scale-out scenarios. For example, Gluster has been successfully deployed in peta-byte size archival scenarios, where the goal was moderate performance in the < \$0.25/GB range. Additionally, we have been deployed in very high performance production scenarios, and have demonstrated throughput exceeding 22 GB/s

¹ Results of scalability testing performed at Gluster running an I/Ozone test on 8 clients connecting to between 1 and 8 server nodes. Total capacity was 13 TB. Network was 1 GbE. Servers were configured with single quad core Intel Xeon CPUs and 8 GB of RAM. Clients had two GB of Ram.

2.0 TECHNICAL DIFFERENTIATORS

There are seven fundamental technical differentiators between Gluster and traditional systems. These are discussed briefly below.

2.1 SOFTWARE ONLY

We believe that storage is a software problem - one which cannot be solved by locking customers into a particular vendor or a particular hardware configuration. We have designed Gluster to work with a wide variety of industry standard storage, networking, and compute solutions. For commercial customers, Gluster is delivered as virtual appliance, either packaged within a virtual machine container, or an image deployed in a public cloud. Within the Gluster open source community, GlusterFS is often deployed on a wide range of operating systems leveraging off the shelf hardware. For more information on deploying the Gluster Virtual Storage Appliance for VMware please see the white paper titled, "How to Deploy Gluster Virtual Storage Appliance for VMware." For more information on deploying Gluster in the Amazon Web Services cloud, please see the white paper titled, "How to Deploy Gluster Amazon Machine Image."

2.2 OPEN SOURCE

We believe that the best way to deliver functionality is by embracing the open source model. As a result, Gluster users benefit from a worldwide community of thousands of developers who are constantly testing the product in a wide range of environments and workloads, providing continuous feedback and support—and providing unbiased feedback to other users. And, for those users who are so inclined, Gluster can be modified and extended, under the terms of the GNU Affero General Public License (AGPL). For more information on the Gluster community, please visit www.gluster.org

2.3 COMPLETE STORAGE OPERATING SYSTEM STACK

Our belief is that it is important not only to deliver a distributed file system, but also to deliver a number of other important functions in a distributed fashion. Gluster delivers distributed memory management, I/O scheduling, software RAID, and self-healing. In essence, by taking a lesson from micro-kernel architectures, we have designed Gluster to deliver a complete storage operating system stack in user space.

2.4 USER SPACE

Unlike traditional file systems, Gluster operates in user space. This makes installing and upgrading Gluster significantly easier. And, it means that users who choose to develop on top of Gluster need only have general C programming skills, not specialized kernel expertise.

2.5 MODULAR, STACKABLE ARCHITECTURE

Gluster is designed using a modular and stackable architecture. To configure Gluster for highly specialized environments (e.g. large number of large files, huge numbers of very small files, environments with cloud storage, various transport protocols, etc.) it is a simple matter of including, or excluding particular modules.

For the sake of stability, certain options should not be changed once the system is in use (for example, one would not remove a function such as replication if high availability was a desired functionality.)

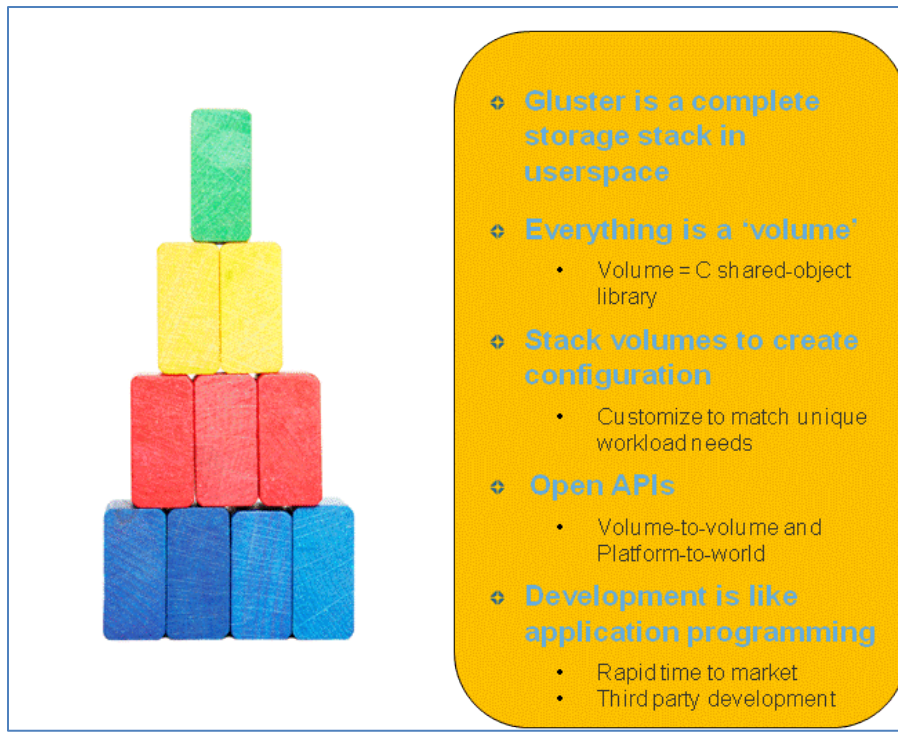


Figure 3 Modular and Stackable Architecture

2.6 DATA STORED IN NATIVE FORMATS

With Gluster, data is stored on disk using native formats (e.g. EXT3, EXT4, XFS). Gluster has implemented various self healing processes for data. As a result, the system is extremely resilient. Furthermore, files are naturally readable without GlusterFS. So, if a customer chooses to migrate away from Gluster, their data is still completely usable without any required modifications.

2.7 NO METADATA WITH THE ELASTIC HASH ALGORITHM

In a scale-out system, one of the biggest challenges is keeping track of the logical and physical location of data (location metadata). Most distributed systems solve this problem by creating a separate index with file names and location metadata. Unfortunately, this creates both a central point of failure and a huge performance bottleneck. As traditional systems add more files, more servers, or more disks, the central metadata server becomes a performance chokepoint. This becomes an even bigger challenge if the workload consists primarily of small files, and the ratio of metadata to data increases.

Unlike other distributed file systems Gluster does not create, store, or use a separate index of metadata in any way. Instead, Gluster locates files algorithmically. All storage system servers in the cluster have the intelligence to locate any piece of data without looking it up in an index or querying another server. All a storage system server needs to do to locate a file is to know the pathname and filename and apply the algorithm. This fully parallelizes

data access and ensures linear performance scaling. The performance, availability, and stability advantages of not using metadata are significant. This is discussed in greater detail in the next section, “The Elastic Hashing Algorithm.”

3.0 THE ELASTIC HASHING ALGORITHM

For most distributed systems, it is the treatment of metadata that most significantly impacts the ability to scale.

In a scale-out system, workloads and data are spread across a large numbers of physically independent storage and compute units. A central problem to solve in such a system is ensuring that all data can be easily located and retrieved.

3.1 CENTRALIZED METADATA SYSTEMS

Most legacy scale-out storage systems address this problem via the use of a central metadata index. As you can imagine, this is a centralized server which contains the names and associated physical locations of all files.

Such a system has two serious flaws.

1. **A Performance Bottleneck:** The metadata server quickly becomes a performance bottleneck. It is the fundamental nature of metadata that it must be synchronously maintained in lockstep with the data. Any time the data is touched in any way, the metadata must be updated to reflect this. Many people are surprised to learn that for every read operation touching a file, this requirement to maintain a consistent and correct metadata representation of “access time” means that the timestamp for the file must be updated, resulting in a write operation to the metadata. As the number of files and file operations increases, the centralized metadata quickly becomes a performance bottleneck.

Serious scaling issues are realized as:

- The number of files increase
 - The number of file operations increase
 - The number of disks increase
 - The number of storage systems increase
 - The average size of the files decreases (as the average file size decreases, the ratio of metadata to data increases.)
2. **A Single Point of Failure:** Perhaps more serious is the fact that the centralized metadata server becomes a **single point of failure**. If the metadata server goes offline, all operations essentially cease. If the metadata server is corrupted, recovering data involves - at best - a lengthy filesystem check (FSCK) operation and - at worst – the data is unrecoverable.

Figure 4, below, illustrates a typical centralized metadata server implementation. One can see that this approach results in considerable overhead processing for file access, and by design is a single point failure. This legacy approach to scale-out storage is not congruent with the requirement of the modern data center or with the burgeoning migration to virtualization and cloud computing.

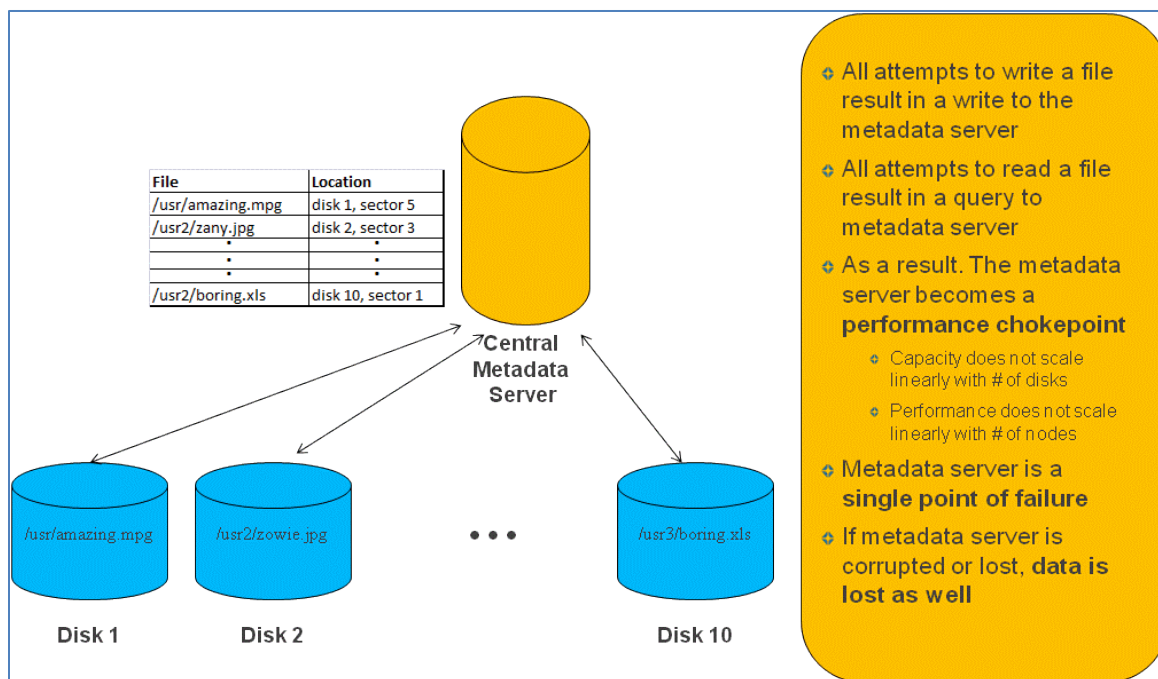


Figure 4 Centralized Metadata Approach

3.2 DISTRIBUTED METADATA SYSTEMS

An alternative approach is to forego a centralized metadata server in favor of a distributed metadata approach. In this implementation, the index of location metadata is spread among a large number of storage systems.

While this approach would appear on the surface to address the shortcomings of the centralized approach, it introduces an entirely new set of performance and availability issues.

1. **Performance Overhead:** Considerable performance overhead is introduced as the various distributed systems try to stay in sync with data via the use of various locking and synching mechanisms. Thus, most of the performance scaling issues that plague centralized metadata systems plague distributed metadata systems as well. Performance degrades as there is an increase in files, file operations, storage systems, disks, or the randomness of I/O operations. Performance similarly degrades as the average file size decreases.

While some systems attempt to counterbalance these effects by creating dedicated solid state drives with high performance internal networks for metadata, this approach can become prohibitively expensive.

2. **Corruption Issues:** Distributed metadata systems also face the potential for serious corruption issues. While the loss or corruption of one distributed node won't take down the entire system, it can corrupt the entire system. When metadata is stored in multiple locations, the requirement to maintain it synchronously also implies significant risk related to situations when the metadata is not properly kept in synch, or in the event it is actually damaged. The worst possible scenario involves apparently-successful updates to file data and metadata to separate locations, without correct synchronous maintenance of metadata, such that there is no longer perfect agreement among the multiple instances. Furthermore, the chances of a corrupted storage system increase exponentially with the number of systems. Thus, concurrency of metadata becomes a significant challenge.

Figure 5, below, illustrates a typical distributed metadata server implementation. It can be seen that this approach also results in considerable overhead processing for file access, and by design has built-in exposure for corruption scenarios. Here again we see a legacy approach to scale-out storage not congruent with the requirement of the modern data center or with the burgeoning migration to virtualization and cloud computing.

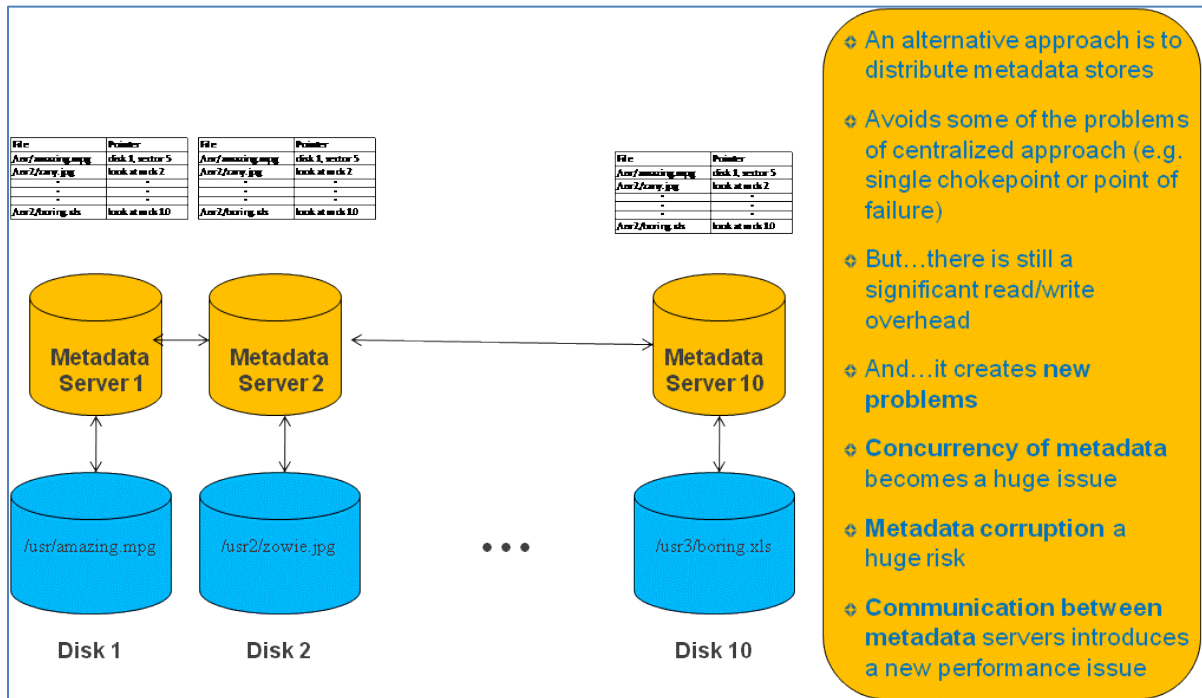


Figure 5 Decentralized Metadata Approach

3.3 AN ALGORITHMIC APPROACH (NO METADATA MODEL)

As we have seen so far, any system which separates data from location metadata introduces both performance and reliability concerns. Therefore, Gluster designed a system which does not separate metadata from data, and which does not rely on any separate metadata server, whether centralized or distributed.

Instead, Gluster locates data algorithmically. Knowing nothing but the path name and file name, any storage system node and any client requiring read or write access to a file in a Gluster storage cluster performs a mathematical operation that calculates the file location. In other words, there is no need to separate location metadata from data, because the location can be determined independently.

We call this the Elastic Hashing Algorithm, and it is key to many of the unique advantages of Gluster. While a complete explanation of the Elastic Hashing Algorithm is beyond the scope of this document, the following is a simplified explanation that should illuminate some of the guiding principles of the Elastic Hashing Algorithm.

The benefits of the Elastic Hashing Algorithm are fourfold:

1. The algorithmic approach makes Gluster faster for each individual operation, because it calculates metadata using an algorithm, and that approach is faster than retrieving metadata from any storage media.
2. The algorithmic approach also means that Gluster is faster for large and growing individual systems because there is never any contention for any single instance of metadata stored at only one location.
3. The algorithmic approach means Gluster is **faster and achieves true linear scaling for distributed deployments**, because each node is independent in its algorithmic handling of its own metadata, eliminating the need to synchronize metadata.
4. Most importantly, the algorithmic approach means that Gluster is safer **in distributed deployments**, because it eliminates all scenarios of risk which are derived from out-of-synch metadata (and that is arguably the most common source of significant risk to large bodies of distributed data).

To explain how the Elastic Hashing Algorithm works, we will examine each of the three words (algorithm, hashing, and elastic.)

Let's start with Algorithm. We are all familiar with an algorithmic approach to locating data. If a person goes into any office that stores physical documents in folders in filing cabinets, that person should be able to find the "Acme" folder without going to a central index. Anyone in the office would know that the "Acme" folder is located in the "A" file cabinet, in the drawer marked "Abott-Agriculture," between the Acela and Acorn folders.

Similarly, one could implement an algorithmic approach to data storage that used a similar "alphabet" algorithm to locate files. For example, in a ten system cluster, one could assign all files which begin with the letter "A" to disk 1, all files which begin with the letter "Z" to disk 10, etc. Figure 6, below illustrates this concept.

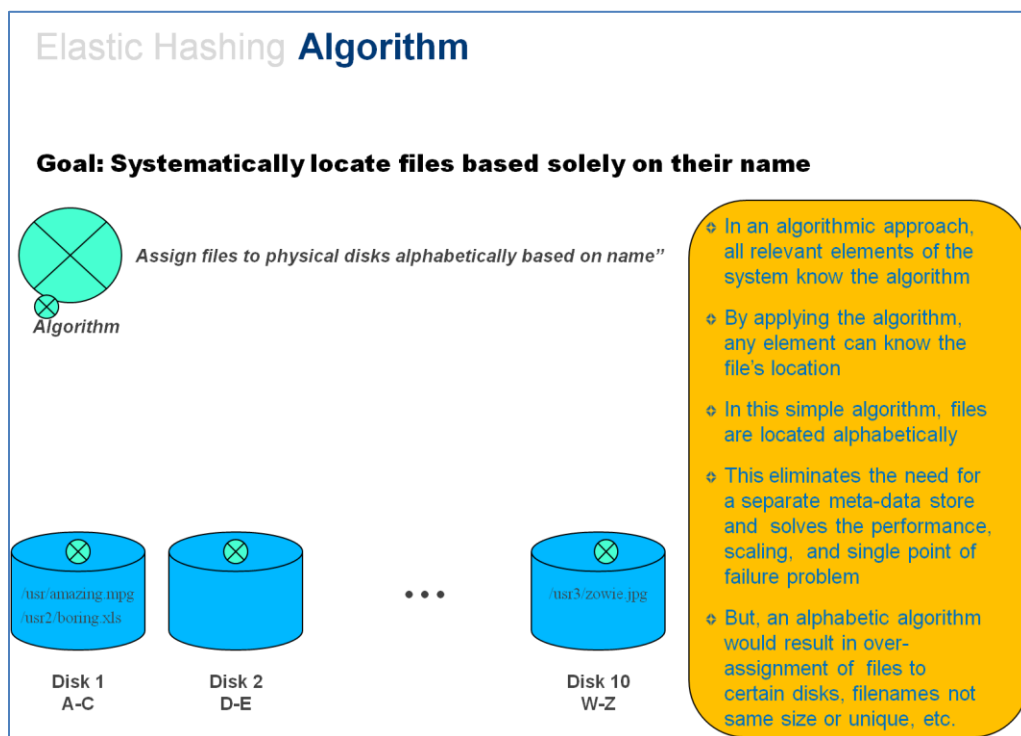


Figure 6: Understanding EHA: Algorithm

Because it is easy to calculate where a file is located, any client or storage system could locate a file based solely on its name. Because there is no need for a separate metadata store, the performance, scaling, and single point-of-failure issues are solved.

Of course, an alphabetic algorithm would never work in practice. File names are not themselves unique, certain letters are far more common than others, we could easily get hotspots where a group of files with similar names are stored, etc.

3.4 THE USE OF HASHING

To address some of the abovementioned shortcomings, you could use a hash-based algorithm. A hash is a mathematical function that converts a string of an arbitrary length into a fixed length values. People familiar with hash algorithms (e.g. the SHA-1 hashing function used in cryptography or various URL shorteners like bit.ly), will know that hash functions are generally chosen for properties such as *determinism* (the same starting string will always result in the same ending hash), and *uniformity* (the ending results tend to be uniformly distributed mathematically). Gluster's Elastic Hashing Algorithm is based on the Davies-Meyer hashing algorithm,

In the Gluster algorithmic approach, we take a given pathname/filename (which is unique in any directory tree) and run it through the hashing algorithm. Each pathname/filename results in a unique numerical result.

For the sake of simplicity, one could imagine assigning all files whose hash ends in the number 1 to the first disk, all which end in the number 2 to the second disk, etc. Figure 7, below, illustrates this concept.

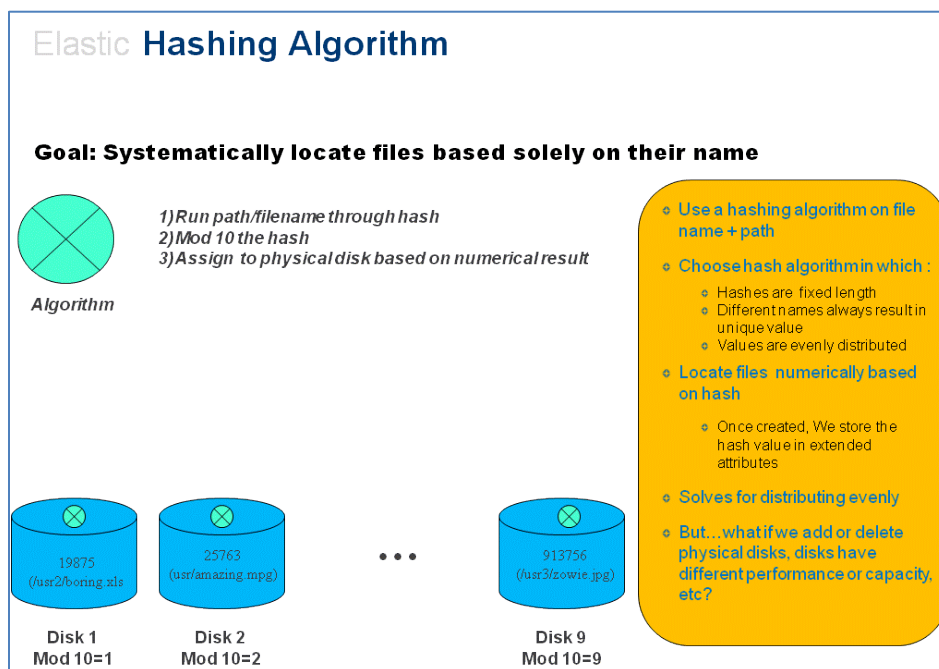


Figure 7 Understanding EHA: Hashing

Of course, questions still arise. What if we add or delete physical disks? What if certain disks develop hotspots? To answer that, we'll turn to a set of questions about how we make the hashing algorithm "elastic".

3.5 MAKING IT ALL ELASTIC: PART I

In the real world, stuff happens. Disks fail, capacity is used up, files need to be redistributed, etc.

Gluster addresses these challenges by:

1. Setting up a very large number of *virtual* volumes
2. Using the hashing algorithm to assign files to virtual volumes
3. Using a separate process to assign virtual volumes to multiple physical devices

Thus, when disks or nodes are added or deleted, the algorithm itself does not need to be changed. However, virtual volumes can be migrated or assigned to new physical locations as the need arises. Figure 8, below, illustrates the Gluster "Elastic Hashing Algorithm".

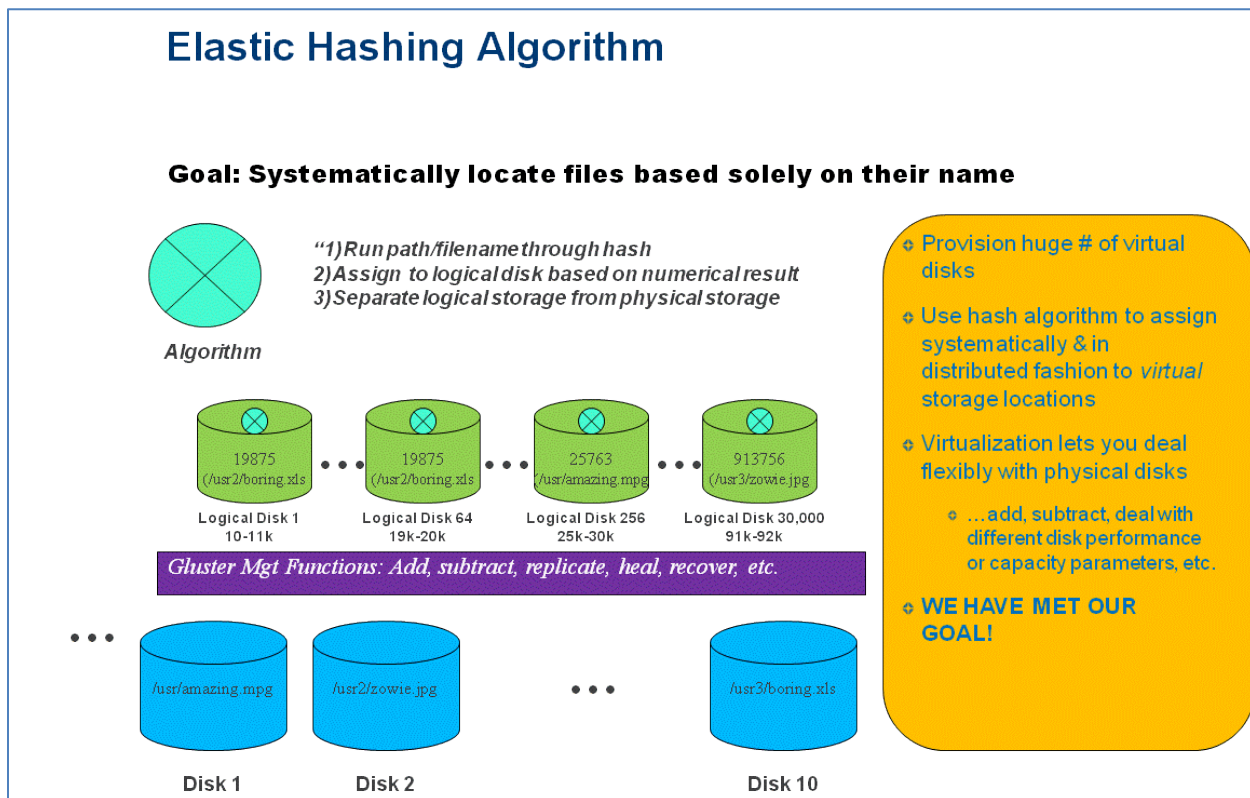


Figure 8 Understanding EHA: Elasticity

For most people, the preceding discussion should be sufficient for understanding the Elastic Hashing Algorithm. It oversimplifies in some respects for pedagogical purposes. (For example, each folder is actually assigned its own hash space.). Advanced discussion on Elastic Volume Management, Moving, or Renaming, and High Availability follows in the next section, Advanced Topics.

ELASTIC VOLUME MANAGEMENT

Since the elastic hashing approach assigns files to logical volumes, a question often arises: “How do you assign logical volumes to physical volumes?”

In versions 3.1 and later of Gluster, volume management is truly elastic. Storage volumes are abstracted from the underlying hardware and can grow, shrink, or be migrated across physical systems as necessary. Storage system servers can be added or removed on-the-fly with data automatically rebalanced across the cluster. Data is always online and there is no application downtime. File system configuration changes are accepted at runtime and propagated throughout the cluster allowing changes to be made dynamically as workloads fluctuate or for performance tuning.

RENAMING OR MOVING FILES

If a file is renamed, the hashing algorithm will obviously result in a different value, which will frequently result in the file being assigned to a different logical volume, which might itself be located in a different physical location.

Since files can be large and rewriting and moving files is generally not a real-time operation, Gluster solves this problem by creating a pointer at the time a file (or set of files) are renamed. Thus, a client looking for a file under the new name would look in a logical volume and be redirected to the old logical volume location. As background processes cause files to be migrated, the pointers are removed.

Similarly, if files need to be moved or reassigned (e.g. if a disk becomes “hot” or degrades in performance), reassignment decisions can be made in real-time, while the physical migration of files can happen as a background process.

HIGH AVAILABILITY

Generally speaking, Gluster recommends the use of mirroring (2, 3, or n-way) to ensure availability. In this scenario, each storage system server is replicated to another storage system server using synchronous writes. The benefits of this strategy are full fault-tolerance; failure of a single storage server is completely transparent to GlusterFS clients. In addition, reads are spread across all members of the mirror. Using GlusterFS there can be an unlimited number of members in a mirror. While the elastic hashing algorithm assigns files to unique logical volumes, Gluster ensures that every file is located on at least two different storage system server nodes. Mirroring without distributing is supported on Gluster clusters with only two storage servers.

While Gluster offers software-level disk and server redundancy at the storage system server level, we also recommend the use of hardware RAID (e.g. RAID 5 or RAID 6) within individual storage system servers to provide an additional level of protection.

For more information, please see “An Introduction to Configuring Gluster” available on our Community web site at <http://www.gluster.org>

4.0 CONCLUSION

By delivering increased performance, scalability, and ease-of-use in concert with reduced cost of acquisition and maintenance, Gluster is a revolutionary step forward in data management. Multiple advanced architectural design decisions make it possible for Gluster to deliver great performance, greater flexibility, greater manageability, and greater resilience at a significantly reduced overall cost. The complete elimination of location metadata via the use of the Elastic Hashing Algorithm is at the heart of many of Gluster's fundamental advantages, including its remarkable resilience, which dramatically reduces the risk of data loss, data corruption, or data becoming unavailable.

To start a functional trial, or download a copy of Gluster, visit us at <http://www.gluster.com/trybuy>

To speak with a Gluster representative about how to solve your particular storage challenges, phone us at +1 (800) 805-5215.

5.0 GLOSSARY

Block storage: Block special files or block devices correspond to devices through which the system moves data in the form of blocks. These device nodes often represent addressable devices such as hard disks, CD-ROM drives, or memory-regions.ⁱ Gluster supports most POSIX compliant block level file systems with extended attributes. Examples include ext3, ext4, ZFS, etc.

Distributed file system: is any file system that allows access to files from multiple hosts sharing via a computer network.ⁱⁱ

Metadata: is defined as data providing information about one or more other pieces of data.ⁱⁱⁱ

Namespace: is an abstract container or environment created to hold a logical grouping of unique identifiers or symbols.^{iv} Each Gluster cluster exposes a single namespace as a POSIX mount point that contains every file in the cluster.

POSIX: or "**P**ortable **O**perating **S**ystem **I**nterface [for Unix]" is the name of a family of related standards specified by the IEEE to define the application programming interface (API), along with shell and utilities interfaces for software compatible with variants of the Unix operating system.^v Gluster exports a fully POSIX compliant file system.

RAID: or "**R**edundant **A**rray of **I**nexpensive **D**isks", is a technology that provides increased storage reliability through redundancy, combining multiple low-cost, less-reliable disk drives components into a logical unit where all drives in the array are interdependent.^{vi}

Userspace: Applications running in user space don't directly interact with hardware, instead using the kernel to moderate access. Userspace applications are generally more portable than applications in kernel space. Gluster is a user space application.

CTDB

CTDB is primarily developed around the concept of having a shared cluster file system across all the nodes in the cluster to provide the features required for building a NAS cluster.

<http://ctdb.samba.org/>

ⁱ http://en.wikipedia.org/wiki/Device_file_system#Block_devices

ⁱⁱ http://en.wikipedia.org/wiki/Distributed_file_system

ⁱⁱⁱ http://en.wikipedia.org/wiki/Metadata#Metadata_definition

^{iv} http://en.wikipedia.org/wiki/Namespace_%28computer_science%29

^v <http://en.wikipedia.org/wiki/POSIX>

^{vi} <http://en.wikipedia.org/wiki/RAID>