Whitepaper

Gluster

File System Architecture

Gluster File System Architecture

Introduction

GlusterFS is a scalable open source clustered file system that offers a global namespace, distributed front end, and scales to hundreds of petabytes without difficulty. By leveraging commodity hardware, Gluster also offers extraordinary cost advantages benefits that are unmatched in the industry. GlusterFS is the core of the integrated Gluster Storage Platform software stack.

In this paper, we discuss the key architectural components of GlusterFS and how these components impact production deployments on a daily basis.

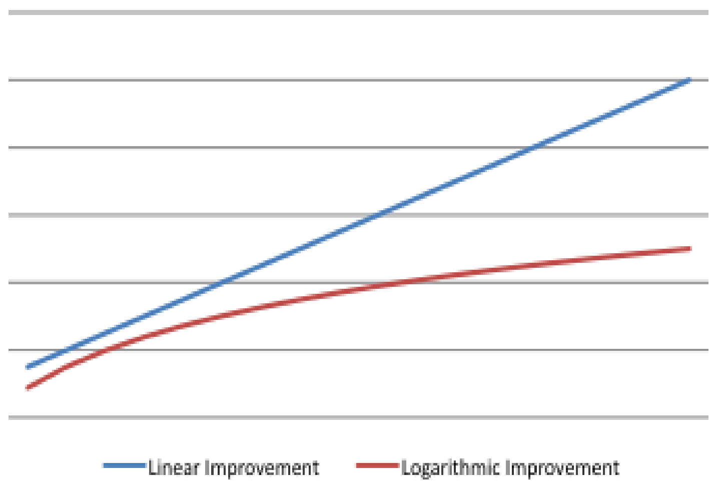
At the heart of the design is a completely new view of how storage architecture should be done. The result is a system that scales linearly, is highly resilient, and offers extraordinary performance. Additionally, Gluster brings compelling economics by deploying on low cost commodity hardware and scaling horizontally and performance and capacity requirements grow. The Gluster Storage Platform integrates GlusterFS with an operating system layer and web-based management and installation tool, simplifying the task of deploying petabyte-scale clustered storage to two steps and just a few mouse clicks.

Scaling Linearly

Storage doesn't scale linearly. This seems somewhat counter-intuitive on the surface 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.

The problem with the current approach in the market is that systems scale logarithmically. With 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.



To overcome this limitation, it is necessary to completely revisit the underlying architecture. Any system that requires end-to-end synchronization of metadata or offers a limited numbers of ingress/egress networking ports must be from its base architecture. Those solutions that cannot act as a cluster of independent storage units are bound to find a scalability limitation sooner rather than later.

The Fundamental Shift

There are three fundamental changes to how storage must be done in order to achieve true linear scalability:

1. The elimination of metadata synchronization and updates.
2. Effective distribution of data to achieve scalability and reliability.
3. The use of parallelism to maximize performance.

The impact of these changes is significant – scalability is achieved while maintaining resiliency. Even better, a well-balanced system also results in improved performance.

To understand how these impacts happen with the three changes listed, let's dig into how these changes work.

Metadata – The Gluster Key to Scalability

One of the most important advantages found in Gluster architecture is its liberation from any dependency on metadata, unique among all commercial storage management systems. This fundamental shift in architecture addresses the core issues surrounding metadata in file systems.

Rather than wrestling with complexities of metadata in a centralized or distributed environment, the Gluster team simply got rid of it.

The result: The unique ‘no-metadata server’ architecture eliminates a bottleneck resulting and provides increased scalability, reliability, and performance.

Background: Why Metadata Matters

For file systems that depend on it, metadata is the heart and soul of how data is organized. Those who are intimately involved with resolving metadata issues are often already familiar the challenges that come with scaling, especially in a distributed environment. Before diving into the nuances of how Gluster works without metadata, let’s stop for a moment and get some background on the topic. Those already familiar with how metadata works may wish to skip the background material.

How Does Metadata Impact Performance?

Gluster routinely demonstrates superior performance in customer deployments, even under circumstances customers expect to severely limit performance. For example, a simple measure of performance involves simply timing how long it takes to *read* or *write* a single large file. That is called “sustained sequential access” and it is well-known to be the easiest thing any file system can be asked to do. Such brute force metrics tell the customer nothing about how the system would perform in a real world environment wherein other tasks are being simultaneously handled, or where the file (large or small) is being accessed piecemeal with random offsets.

The more complicated the workload, the more you would observe metadata also being exercised equally or greater in proportion to the number of I/O events directed toward the contents of each targeted file. This issue especially comes to the fore in an extremely significant way if the file system being tested has been replicated or otherwise distributed so that users can simultaneously access the same data sets with complicated random access patterns, all being tested in the context of permissions and other attributes contained in the metadata associated with each and every file and directory or subdirectory (which by definition also must be replicated and distributed).

Also consider that an additional layer of complexity implies overlapping layers of mixed latency among the remote file system clients being used by the end-users, such that their workload is arriving via high-performance LANs *and* across high-performance WANS (slower than LANs) *and* across much-slower networks (sluggish LANs or narrow-pipe WANs) in a variety of settings distributed geographically — creating inherent complexity for the distributed metadata updates as well as the processes by which any updates to the data itself must be carefully applied so as to be inherently correct (i.e., with chronological sequence preserved).

Such simplistic, brute force one-large-file Read or Write benchmarks begin to pale by comparison to the real world or by comparison to more subtle and refined metrics — as well as the especially challenging nature of actual real world workloads experienced commonly by robust solutions like Gluster, as deployed daily all over the world.

One big reason Gluster performs so extraordinarily well in actual customer deployments and across a wide variety of demanding benchmarks where real world workloads are more realistically simulated is simple: Gluster does not have a bottleneck regarding metadata. In fact, it does not need to scale its handling of metadata at all! Let’s examine how and why that is the case.

What are the Potential Architecture Bottlenecks Associated with Metadata?

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, incurring a *write* operation. Yes, there are optimization techniques involving ways to collect these together and apply them in sets, but nevertheless, the more “expensive” (consumptive of CPU and other system resources) *write* operations associated with maintaining access time metadata are an overhead to *read* operations. And that is merely one example selected because it is simplest to understand.

Where this becomes especially onerous is the requirement to synchronously maintain a current and consistent version of metadata for every file even when there are multiple instances of the file spread across multiple nodes. This is common with modern storage systems because of the need to distribute and share data among users at multiple sites which are often geographically dispersed — in New York, USA, and New Delhi, India, perhaps.

The more nodes there are, in a distributed deployment, and the more they are geographically dispersed, the worse this overhead problem can affect the performance of the system. As the amount of data grows there is a corresponding increase in the metadata and the overhead required to manage it. And even in the simplest such arrangement with the greatest opportunity for performance optimization — two sites, with one a mirror image of the other (intended for Disaster Recovery and Business Continuance) separated by only a few miles of dedicated network connectivity — this bottleneck can throttle performance and multiply the cost of the solution significantly. (The Disaster Recovery site might be designed to have slower storage at a lower price point, but the need to wait on synchronous metadata updates can result in application performance degradation at the primary site!)

What about cutting corners and only focusing on metadata consistency at one location, and allowing "drift" at the others in a distributed architecture? Unfortunately, this corruption of the stored data image has serious implications: for Disaster Recovery, the lost or inconsistent metadata could make the data itself useless in the event that the primary has been destroyed in a natural disaster or other catastrophe. In a multi-node distributed architecture, where it is necessary to have a half-dozen or more file servers to meet the sheer I/O workload demand of a set of compute servers, it would be impossible to dynamically load balance if every time the task was mapped to a particular image of the data it "appeared" to be different (different file names, different permissions, different access times — any or all of these attributes are often critical to the behavior of a software application and can affect the outcome).

In other words, there are a great many good and inescapable reasons why traditional storage vendors who do not have the advantages of the 'no-metadata' architecture accept the burdensome loss of performance implied by the need to maintain multiple images of every file's metadata at each location where the file exists. And performance is only one side of the "metadata burden" coin; the other is risk.

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. How can this "split brain" problem be reconciled? One could be trusted and the other overwritten. If there are three or more, voting can take place to establish a quorum, but even this method breaks down if there are three or more apparently correct outcomes.

Bottom line: the more instances, the more risk. The more versions, the less likely a simple reconciliation can be attempted or implemented.

Gluster Has No Metadata

The architecture within Gluster does not depend on metadata in any way. Therefore, Gluster has no performance bottlenecks and no inconsistency risks as described above, related to metadata. Instead of applying complicated workarounds in the manner of traditional file system suppliers and vendors who seek to mitigate the problem with complex optimizations and shortcuts, Gluster instead eliminates the root cause entirely.

How can Gluster avoid metadata? For all files and directories, instead of storing associated metadata in a set of static data structures (whether replicated and optionally distributed, or kept locally) — and also instead of applying the same inadequate band-aid to address the classic "metadata bottleneck" problem encountered by traditional distributed file system models, by moving metadata into a dedicated server with its own bottlenecks and issues — Gluster instead generates the equivalent information on-the-fly using algorithms. The results of those calculations are dynamic values acquired wherever needed in each of one of more nodes in a Gluster deployment. This eliminates the risk that metadata will ever get out of synch because the algorithms are universal and omnipresent across the distributed architecture, and therefore for many fundamental reasons simply cannot ever be out of synch. And the implications for performance are staggering.

Gluster can process all data access operations independently at all locations throughout the distributed architecture because there is no requirement for the nodes to “stay in synch.” That allows for *linear scaling with no overhead*. “Linear Scaling” is a much-abused phrase among storage industry participants. 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).

The above paragraphs may seem like straightforward observations which ought to apply in many situations beyond Gluster, but they do not. 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.

And even if those additional tasks could be done with near-zero “effort” (in the CPU and other system-resource sense of the term), the latency resulting from waiting for the responses across the networks connecting the distributed nodes in those traditional system architectures will always impact performance scaling proportionally relative to the speed and responsiveness of the networking connecting the nodes to each other (which itself is affected by their locality or remoteness in physical placement, and the choice of the transport layer for the network connectivity (e.g., highly-efficient via RDMA crossing 2 meters of QDR InfiniBand; or inefficiently achieved using simple TDM competition for a narrow pipe bouncing off a satellite link). This is why claims of linear scalability often break down for traditional distributed architectures.

Gluster, however, quite simply scales in a truly linear manner because there is absolutely no reason for it not to do so. Also, by not depending on static data structures (whether single-instance or replicated throughout a distributed deployment), Gluster is also faster in absolute terms for two reasons: configurability; and the proportional reduction of I/O events associated with handling any application-level file system workload. What does that mean?

Configurability is a long word for a simple concept. Basically, if you change the hardware you are changing the configuration. Gluster is not bottlenecked by a dependence on metadata, and therefore a bold claim is possible to make here. Regardless of the type of workload presented by the NFS, CIFS, and local application software instances demanding I/O action from Gluster, you can achieve twice the performance by doubling the CPU and disks. Triple the performance by tripling the hardware. Quadruple it by quadrupling the hardware — and so on. That control over performance via configuration is only possible with Gluster and not with traditional file systems because of the issue of metadata.

An example will make it very clear. The other traditional approaches might be given a 10x increase in hardware resources, but only see a 5% performance increase apparent to the application (this is not uncommon. It happens all the time, in all manner of application software.) That is because most of the additional hardware idles in a “wait state” while the system as a whole suffers with contention for access to metadata (whether to read it or update it to maintain it in synchronous correctness). Gluster has no such contention because each element can independently calculate all metadata for itself with its own independent instance of the algorithms which are universal.

The other performance benefit does not concern a large system trying to scale up to be larger. If given a data server of finite size, consider how much performance it can deliver if any of its valuable and finite resources are tied up in a “wait state” (even if there is no contention involved) because the metadata operations execute slowly, limited by the speed at which the I/O subsystem is able to respond. With disks, this can be a very long time — multiple milliseconds for CPUs running software at multi-gigahertz clock rates. Even when the storage is faster than rotational media, as is the case with solid state storage solutions or flash disks, it is still slower than a multi-gigahertz CPU can complete an algorithmic calculation that would only consume a few microseconds.

Bottom Line: Gluster is Faster, Achieves True Linear Scalability, and is Safer

- Gluster is faster for each individual operation because it calculates metadata using algorithms and that approach is faster than retrieving metadata from any storage media.
- 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.

- 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.
- 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).

Distribution of Data: How the Replication Works

The placement of data has been a challenge ever since there was a place to store data in the first place. The balance between scalability and performance is a difficult one to maintain, especially when making sure that the result is resilient to events such as hardware failure or power loss.

Traditionally, metadata tells us where the actual data is stored but as noted in the previous section, Gluster has moved away from metadata and instead uses an algorithm to determine what the metadata looks like in real-time. Using this algorithm, file distribution is intelligently handled and has the benefit of making sure that files larger than a single storage server are correctly handled.

To understand how the distribution of data and corresponding replication works in Gluster, we turn to understanding the algorithm that drives it.

The Elastic Hash

The Elastic Hash is a new storage-specific algorithm developed by the Gluster team that is responsible for determining where the data is kept and is key to the ability to function without metadata.

At the heart of the implementation are two key features:

1. The algorithm can account for the addition and removal of servers without losing track of where existing data is placed.
2. The algorithm works at a file level instead of a block level.

The ability to account for the addition and removal of servers is atypical of this class of algorithms. Most implementations rely on the fact that the number of places data can go remains constant. In storage, the addition of storage is typically handled through abstraction layers (logical volumes) so that specific file systems do not need to be altered; rather, multiple file systems are glued together using metadata to track where everything is. As discussed in the previous section, metadata simply does not have the scalability needed over the long term.

The second feature is significant: the move to operating at a file level instead of a block level combined with the removal of metadata gives Gluster the capacity to scale to extraordinary levels.

Traditionally, file systems work with individual blocks on disks and count on the presence of RAID to handle individual block failures. Network partitions or the ability to simply take a batch of disks offline (say, to move them from one rack to another) is not dealt with at all. If disks go offline, data is unavailable. If the failure happens on multiple disks in the same RAID array, it is possible to lose all of the data on the file system even if the other disks are fine. Gluster does not have this problem.

The Elastic Hash makes data placement decisions at a host level and then places complete files on its selected storage. If configured by the administrator, multiple copies of the complete file can be placed on multiple hosts. This means hosts including all of their associated disks, can go offline for any reason (e.g., a network partition, or even just maintenance) and other hosts are able to serve up complete copies of the file.

Elastic Hash takes that one step further and optimizes selection based not just on availability, but performance as well. Thus, if there is a copy of the file that is closer to the user, the closest copy is used. This feature creates a natural replication behavior since the cluster of storage hosts can be spread across two sites with copies of data located at both sites while letting users select the site that is closest to them for file access.

In the event the storage cluster experiences permanent damage, e.g., a host experiences a major disk failure that renders its local file system unreadable, the replacement host can join the cluster and the healing process will occur automatically. This allows the storage administrator to delegate basic system upkeep without having to concern himself with availability in the event hosts fail.

Parallelism

In earlier sections we discuss the impact of removing metadata and the Elastic Hash algorithm on data placement to data integrity and availability. There is, however, one more beneficiary: Performance.

The architectural decision to operate at a host level instead of at an individual file system or block level has allowed Gluster to scale across many CPUs and better distribute the workload of running a storage network over a well balanced mix of server CPUs and their corresponding disks. Along with each CPU is a network connection, data and memory bus, etc. In short, all of the limitations that we identified as being critical flaws in traditional storage's ability to scale are inherently solved with the Gluster architecture.

The highly parallel architecture of Gluster is differentiated from typical networked storage in that there is a far more *intelligent* relationship between available CPUs and spindles. With more intelligent spindles comes greater throughput. Traditional approaches by comparison place many more spindles per networked head. This puts a physical limitation on the collective throughput, especially when taking into consideration the overhead of metadata synchronization.

With even distribution of work across multiple hosts and the use of algorithms to remove the need for metadata, Gluster offers *true linear scalability*. This means five servers deliver 5x the performance of a single server, not just the 2-3x typical of other storage solutions. As storage scales up to petabytes, the capability to linearly scale also means that unlike other solutions, there are no points at which administrators need to consider forklift upgrades to higher end models in order to support higher capacities – Gluster supports a handful of servers is the same manner it does for hundreds of servers.

Reliability

Reliability, like performance and scalability, is not something that can be added as a feature to a storage system as an afterthought. It must be baked into the design from the start. Gluster takes reliability into account from its core design.

To achieve this kind of reliability, Gluster applies several core design principles:

Assume Failure

Failure happens. Hardware fails, disks fail, networks get segmented, and administrators mistakenly trip over power cords. Rather than treat this as the exceptional case, Gluster designs into its core the ability to self-heal and gracefully cope with the failure without requiring administrator intervention.

Don't Redo What Is Already Trusted

There are a number of disk file systems available with existing operating systems that are well tested and in some cases, literally in production use across millions of servers worldwide. Gluster does not seek to replace those. Rather, Gluster stands on the shoulders of giants and leverages their field-proven reliability. Besides providing extraordinary resilience, using well proven disk file systems on existing hosts means that administrators are able to look into each host even without the Gluster layer and see all of their files.

This is a completely new level of freedom for the storage administrator. Gluster actually makes it easy to stop using Gluster. Rather than lock-in administrators into a specific file system that requires proprietary software to access, Gluster takes the approach that it must prove its value on an ongoing basis or the administrator will stop using it – after all, the administrator can get to his data unencumbered.

This is not a bug or a product management oversight; this is design principle.

Don’t Rely on Metadata

Earlier you saw how Gluster doesn’t rely on metadata in order to provide performance. What this architecture also achieves is stellar reliability – without metadata, there is no need to worry about metadata corruption or complex error-prone synchronization events. When a process is made as simple as it can be made, the risk of errors is significantly diminished.

Summary

The Gluster Storage Platform is a revolutionary step forward in data management on every axis and in every dimension: absolute performance; scalability of performance and capacity; manageability at scale; ease of use; reduced cost of acquisition, implementation; daily operation, and per-terabyte for any particular level of desired redundancy or achieved-reliability. The complete elimination of metadata is at the heart of many of its fundamental advantages, including its remarkable resilience, leading to its reduced risk of data loss or data corruption down to conditional states near absolute zero by statistical calculation or logical extrapolation.

In a typical environment, a low-cost implementation will protect the data with measures applied only locally, and Gluster excels in this regard because it can allow *write* operations to be performed independently to two disk targets with no synchronous metadata complications requiring that the two be “compatible” or even similar.

Gluster brings completely new technology that delivers on a wholly new philosophy for storage: the focus is on the compute host, not on the disk drives and shelves. This implies a new level of independence which guarantees new levels of performance, scalability, manageability, and reliability never before seen in any other storage system, with opportunities for integration at the application level — e.g., for virtualization and cloud deployments — which are only now beginning to be better understood by customers and integrators using Gluster.

Gluster — data management for the 21st century, leaving the past behind.