Indian J.Sci.Res. 17(2): 111 - 117, 2018 ISSN: 0976-2876 (Print)
ISSN: 2250-0138(Online)

AN INSIGHT ABOUT GLUSTERFS AND ITS ENFORCEMENT TECHNIQUES

[1] Manikandan Selvaganesan,[2]Mohamed Ashiq Liazudeen,[3] BhuvaneshwariSelvaganesan
[1,2] Associate Software Engineer, Department of Products & Technologies,Red Hat, Bangalore.
[3]Assistant Professor, Dept. of Computer Science and Engg, Sri Eshwar College of Eng, Coimbatore

Abstract—The world keeps contributing to the increase in data everyday drastically. Scientific applications, weather forecasting, researches, hospitals, military services are few such major contributors. As the amount of data increases, the need to provide efficient, easy to use solutions has become one of the main issues for these type of computations. The best solution to this issue is the use of Distributed File Systems(DFS). Some existing Distributed File Systems are too complex to deploy and maintain, although they are extremely scalable and cheap since they can be entirely built out of commodity Operating System(OS) and hardware. GlusterFS solves this problem. Gluster File System(GFS) is open source and is capable of scaling to several petabytes(upto 72 brontobytes) and handling thousands of clients. It is based on a stackable user space design and can deliver exceptional performance for diverse workloads. GlusterFS is written in user space which uses FUSE(Filesystem in user space) to hook itself with the VFS layer. It takes a layered approach to the file system, where features are added/removed as per the requirement. Enforcer is a major component in Gluster which finds its extensive use in production. Enforcer is concerned more about limits. It helps to restrict the usage(both in terms of size and count) at directory or volume level in the file system. In this paper, we discuss the internal working of the translators and key architectural components of GlusterFS. Along with that, one can gain a deep insight about the enforcement techniques, few challenges,undergoingresearchesandprojectsofGlusterFS.

Keywords:Distributed File System(DFS), Gluster File Sys- tem(GFS), Translators, Enforcer, Marker

I. 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. It also offers extraordinary cost advantages benefits that are un- matched in the industry. No longer are users locked into costly, monolithic, legacy storage platforms. GlusterFS gives users the ability to deploy scale-out, virtualized storage scaling from terabytes to petabytes in a centrally managed and commoditized pool of storage,which is available to users in a single mount point, making it simple for the user.

At the heart of the design Gluster File System 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 as perfor- mance and capacity requirements grow.

A. NotonlyStoragebuttheStorageSystem

Storage does not scale linearly. One can think this is counter-intuitive on the surface since it is easy for someone topurchaseanothersetofdiskstodoublethesizeofavailable storage. An important limitation in doing so is that the scalability of storage has multiple dimensions, capacitybeing one of them. Adding capacity is only one dimension, there are few other factors which contribute as well such as the CPU capacity, the scalability of the file system to support the total size. Another major thing is 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. To be precise, as the title says, it is not the storage that needs to scale as much as it is the complete storage system that needs toscale.

B. TraditionalApproach

With current Distributed File Systems(DFS) the problem is that systems scale logarithmically as discussed here[1]. This is re-factored in Gluster file system. With the former approach, storage's useful capacity grows more slowly as it gets larger. This is due to the increased overhead necessaryto maintain data strength. This limitation is examined bytesting the performance of some storage networks which clearly reflects that larger units offer slower aggregate performance than their smaller counterparts. It is necessary to completely revisit the underlying architecture to overcome this limita- tion. Any system that requires end-to-end synchronisation of metadata or offers a limited number of networking ports must be implemented efficiently 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 thanlater.

[1]Corresponding Author

C. True Linear Scalability inGlusterFS

To achieve true linear scalability, there are some funda- mental changes to how storage must be done

- Metadata synchronisation and updates could be elimi- nated.
- The way data is distributed to achieve scalability and reliability.
- Makeuseofparallelismtoimproveperformance.

The impact of these change can significantly result in im- proved performance. GlusterFS is one such proven example.Tounderstand how these are achieved with GlusterFS, let's dig into how these changes work.

II. An Overview OfGlusterFS

As mentioned above, GlusterFS has no metadata server and is capable of linear scaling and can handle uptothousand clients.Let'slookabitmoredeeperonhowarethesefeatures affect performance of any distributed file system and howare they efficiently achieved inGlusterFS.

A. Meta Data's Impact onPerformance

For distributed file systems, metadata is the heart and soul of how data is organized. Scaling and metadataare way dependent. A simple measure of performance involves simply timing of how long it takes to read or write a single large file. This brute force technique is called"sustained sequential access" and which is the basic thing any file system would be expected to do. Anyways that does not tell the customer about how the system would perform in a real world environment. The more complicated theworkload, the more you would observe metadata also being exercised equally or greater in proportion to the number of I/O events directedtowardsthecontentsofeachtargetedfile.

An additional layer of complexity comes with a vari- ety of settings distributed geographically, creating inherent complexity for the distributed metadata updates. It isthe fundamentalnatureofmetadatathatitmustbesynchronously maintained in lockstep with the data. Any time the data is touched in any way, the metadata must be updatedto 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 writeoperation.

Another important issue is that when metadata is stored in multiple locations, the requirement to maintain itsyn- chronously also implies significant risk related to situations when the metadata is not properly kept in sync, or in the event if it is actually damaged. Gluster does not have a bottleneck regarding metadata. In fact, it does not need to scale its handling of metadata at all. Let's look on how the conceptof"NoMetadata"helpsGlustermorescalable.

B. Gluster'sKeys toScalability

One of the most important advantages found in Glus- ter'sarchitecture is its liberation from any dependency on metadata, unique among all commercial storagemanagement systems. This fundamental shift in architecture addresses the core issues surrounding metadata in file systems.Gluster got rid of the complexities of metadata in a centralized or distributedenvironmentwhichotherfilesystemshave.

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) 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 intoa 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 or more nodes in a Gluster deployment. This eliminates the risk that metadata will never get out of sync because the algo- rithms 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 arestaggering.

Gluster can process all data access operations indepen- dently 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.

In the next section, let's look on how GlusterFS is imple- mented using various translators and types of volumes that GlusterFS has, followed by the overall working architecture.

III. Architecture OfGlusterFS

Let's now explore how GlusterFS is implemented in the real world. Brick is the basic unit of storage in GlusterFS, represented by an export directory on a server in the Trusted Storage Pool(TSP). Volume is the collection of bricks and most of the gluster file system operations happen onthe volume. Depending on the need one can choose the type ofvolume.FollowingarethetypeofGlusterFSvolumes:

A. VolumeTypes

- Distributed GlusterFSVolume - This is the default glusterfs volume i.e, while creating a volume if you do not specify the type of the volume the default option is to create a distributed type of volume. Herefiles are distributed across various bricks in the volume. Referring Fig. 1, file1 may be stored only in brick1 or brick2 but not on both. Hence there is no data redundancy.Thepurposeforsuchastoragevolumeisto easily scale the volume size. However this also means that a brick failure will lead to complete loss of data and one must rely on the underlying hardware for data lossprotection.

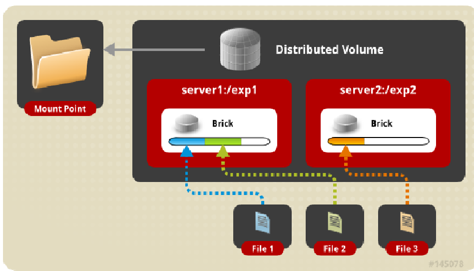

Fig. 1. Plain distribute volume

- Replicated GlusterFSVolume - In this volume we overcome the data loss problem faced in the distributed volume. Here exact copy of the data is maintained on all bricks. The number of replicas in the volume can bedecidedbyclientwhilecreatingthevolume.Onemajor advantage of such a volume is that even if one brick fails the data can still be accessed from its replica brick. Such a volume is used for better reliability and data redundancy.
- Gluster also has other volume types likeDistributed Replica, Striped, Distributed Striped GlusterFSVolumes which has its own additional advantages and usecases. Also, the volume type can be EC(Erasure Coding) or a tiered volume or a disperse volume and so on. The disperse translator is a new type of volumefor GlusterFS that can be used to offer a configurable level of fault tolerance while optimizing the disk space waste. It can be seen as a RAID5-like volume. To avoid triplication(three way replication) which is expensive and instead of wasting two redundant disks for every data, one can use erasure volume[2] which desires protection from double failure. To efficiently access(recently accessed) the files, one can use the tiered volume which contains hot tier and cold tier where the recently accessed ones are stored in the former for a particular timeout(user formattable) and then is moved to the later. Choosing the volume type is based on the use case while deploying in the real world. Letusquicklymoveontothenextaspect(s).

Let's dig more on how Gluster is implemented in the userspace in next section.

B. GlusterFS - User SpaceFile System

GlusterFS is a userspacefilesystem. Being a userspacefilesystem, to interact with kernel VFS, GlusterFS makes use of FUSE(Filesystem in Userspace)[3].

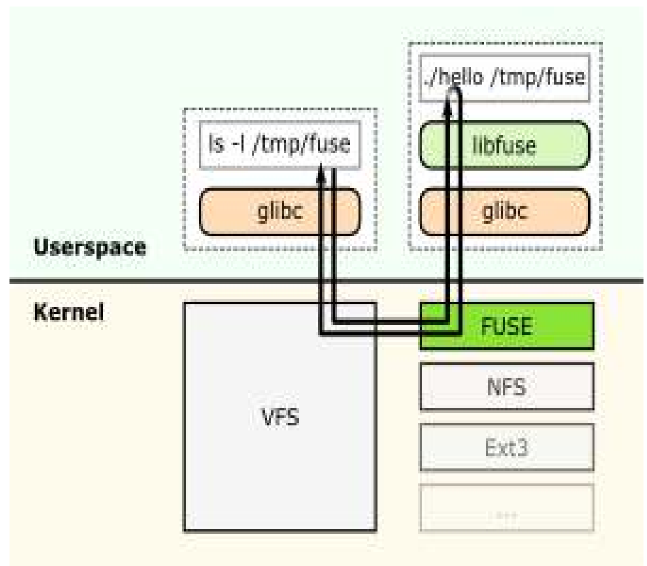

Fig. 2. Structural diagram of FUSE

Fig. 2 shows a filesystem "hello world" that is compiledto create a binary "hello". It is executed with a filesystem mount point /tmp/fuse. Then the user issues a commandls -l on the mount point /tmp/fuse. This command reaches VFS via glibc since the mount /tmp/fuse corresponds to a FUSE based filesystem, VFS passes it over to FUSE module. The FUSE kernel module contacts the actual filesystem binary "hello" after passing through glibc and FUSE library in userspace(libfuse). The result is returned by the "hello" through the same path and reaches the ls -l command. The communication between FUSE kernel module and the FUSE library(libfuse) is via a special file descriptor which is obtained by opening /dev/fuse. This file can be opened multiple times, and the obtained file descriptor ispassed to the mount syscall, to match up the descriptor with the mounted filesystem. Now, we clearly know that GlusterFS incorporatesalotoffeatures.Thesefeaturesareimplemented as translators. Though there are lot of translators, let's gain some insight on cluster and feature translators in the next section.

C. Translators inGlusterFS

A translator converts requests from users intorequests for storage. A translator can modify requests on the way through : convert one request type to another(during the request transfer amongst the translators) and modify paths, flags or even data(e.g. encryption). Translators can intercept or block the requests(e.g. access control) or spawn new requests(e.g. pre-fetch). Translators make use of shared ob- jects to communicate with one another. They are dynam- ically loaded according to volfile. Each translator sets up pointers to parent/child translators, call init(constructors) and call IO function through fops and has the provision for validating/passing options to one another. The configuration of translators(since GlusterFS 3.1) is managed through the gluster command line interface(cli), so one need not know in what order to graph the translators together. Every translator has it'sown functional purpose. Throughout the translators, it's been noted that "Extended Attributes" are important. Let'slookmoreonthisinamoment.

D. ExtendedAttributes

GlusterFS makes use of extended attributes in replication, distribution, striping etc. In DHT a directory must be present on all bricks and each directory copy will be assigned a hash range stored in its extended attribute - trusted.glusterfs.dht. A directory lookup will return the layout(hash ranges collected from the xattrs) which is stored in a table. This helps us to look for missing hash ranges(possible if the brick is down), overlaps, etc. In AFR the extended attribute - trusted.afr.* where * is the brick name, is used for recording operation failure. Consider two bricks brick0 and brick1 in a volume. A file on brick0 has the xattr trusted.afr.brick1 and a file on brick1 has the xattr trusted.afr.brick0. This is because if we store both the state of operation(success or failure) and the actual operation on the same brick and if that brick goes down, then there would be no way to recover from failure

sincewelosethestateoftheoperations.Hencetheoperationa ndthestateofoperationarestoredattwodifferent places. The xattr works as a counter and records counts for three different kinds of operations data, metadata and entry. To perform an operation there are three stages:

1) Preop- whenever a modification is to be made all the counters will be incremented. 2) Op - here the operation is actually performed. 3) Postop - if the operation was successful then the counters are decremented. If the operation wassuccessful across all the bricks then all counters would go back to zero. However in our example if the brick0 was down or had crashed before the operation was successfully completed then the counter for brick0 stored on brick1 willremain non-zero which implies that the operation on brick0was unsuccessful. Now comes the feature translators. Let's look on how quota and marker handles the extended attributes. Quota and marker makes uses of xattr's such as : 1) size - to store the size of directory(or subdirectory). 2) contri - how much(size) of data is being contributed to the ancestor(s). 3) dirty-aflagtomakesuretheatomicityofoperations.

Some of the other xattrs are trusted.gfid used to detect duplication in inode numbers. trusted.glusterfs.test stored in the root directory of every brick used for determiningif xattrs are supported. Native access mechanism being used in GlusterFS is Fuse. GlusterFS also has alternate access mechanisms[5] libgfapi, GlusterNFS, GlusterFS and NFS- Ganesha integration and using Samba(for windows environ- ment) areimplemented.

E. Overall Working ofGlusterFS

Let's consider GlusterFS is installed on a servernode. As soon as it is installed, a gluster management dae- mon(glusterd) binary will be created. This daemon should be running in all participating nodes in the cluster. Afterstarting glusterd, a Trusted Server Pool can be created consisting of all storage server nodes(TSP can contain even a single node). Now bricks which are the basic units of storage can be created as export directories in these servers. Any number of bricks from this TSP can be clubbed together to form a volume. Once a volume is created, a glusterfsd process starts running in each of the participating brick. Along with this, configuration files known as vol files will be generated inside/var/lib/glusterd/vols/.Therewillbeconfigurationfile s corresponding to each brick in the volume. This will contain all the details about that particular brick. Configuration file required by a client process will also be created. Now the file system is ready to use. We can mount this volume on a client machine very easily. When we mount the volume in the client, the client GlusterFS process communicates with the server's glusterd process. Server glusterd process sends a configuration file(vol file) containing the list of client translators and another containing the information of each brick in the volume with the help of which the client GlusterFS process can now directly communicate with each bricks glusterfsd process. The setup is now complete and the volumeisnowreadyforclient'sservice.

As shown in Fig. 3, when a system call(File operation orFop)isissuedbyclient(orapplication)inthemounted

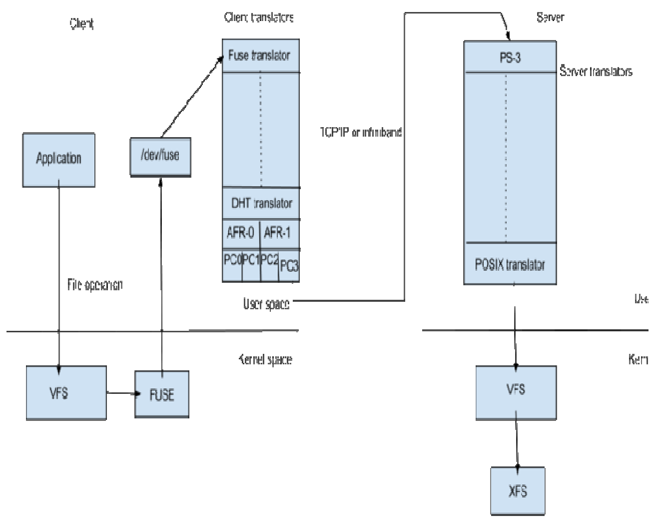

Fig. 3. Overall working of GlusterFS

filesystem, the VFS(identifying the type of filesystem to be glustefs) will send the request to the FUSE kernel module. TheFUSEkernelmodulewillinturnsenditto theGlusterFS in the userspace of the client node via /dev/fuse(this hasbeen described in FUSE section). The GlusterFS process in client consists of a stack of translators called the client translators which are defined in the configuration file(vol file) sent by the storage server glusterd process. In the next section let us discussmoreabouthowquotaenforcesandmarkermaintains accounting and communicate with other translators. Along with that, we will also explore on the techniques used by them.

IV. Enforcer Techniques And The Accounting Translator -Marker

As mentioned in the previous section, quota does the the enforcing works and marker is the one which does the actual accounting. Quota enforcement is done on server side. It is not done in client side, because all the clients could not be trusted and to reduce the traffic on the network(if it is done on the client side, then we have to rely on lookup calls on file/directory(inode) to update the contribution). One can set the quota at the following levels :

- Directorylevel-limitstheusageatthedirectorylevel.
- Volume level - limits the usage at the volume level.

Usercansetbothhard-limitandsoft-limits.

- Soft limit : Messages will be logged on reaching the softlimits.
- Hard limit : Writes will fail with EDQUOT (Disk quota exceedederror)oncetheyreachhardlimit.

Translators marker and enforcer sits on server side

Quotad is a client process without a mount point. It has the same graph as client graph(client->dht->afr->brick). In case of replica we take the one which shows higher size. We store inode-memory(limit) in inode context itself. Write operation is expensive if for every operation we contact quota daemon. So we have stored limit in inode memory as well. Accounting is done using the marker translator. Marker translator is present in each brick of the volume. Accounting process happens in the background and it doesnthappen while the file operation is being carried out simultaneously. Every time a delta value is calculated(the difference between the current size and older size) and then it is sent recursively upwards to the root of the volume, which is also stored as contri in every file/directory. During the initial lookup, extended attributes are created or whenever quotaenable happens, a lookup is initiated and the extended attributes arecreated.

A. QuotaAttributes

Extended attributes of quota[6] look like this :

- trusted.glusterfs.quota.00000000-0000-0000-0000-000000000001.contri=0x000000000000000000000000
  0

  0000000000000000000001
- trusted.glusterfs.quota.limit-set=0x0000000000500 000ffffffffffffffff(5.0isthehardlimitthatssethere)
- trusted.glusterfs.quota.size=0x00000000000000000000 000000000000000000000000001

In the above example:

- ’contri’ xattr: ’contri’ xattr contains the size of how much it contributes to the parent directory(recursive upto every directory upto root). Here, the value ’00000000-0000-0000-0000-000000000001’ before the .contri in thexattr corresponds to the gfidof the parent. About the value ’0x000000000000000000000000000000000 00000000001’, ’0x’ represents hexadecimal and the first sixteen digits(or 64bits) corresponds to the actual size contributed, the next sixteen digits corresponds to the file count and the next sixteen digits corresponds to the dircount.
- ’limit-set’ xattr: The ’limit-set’ xattr, as the name says, helps to enforce(restrict) the usage on a specified directory. trusted.glusterfs.quota.limit-set=0x0000000000500000ffffffffffffffff(5.0 is the hard limit that is set here by the user) In the ’limit-set’ xattr the first sixteen digits corresponds to hard

limit and nextsixteendigitscorrespondstosoftlimit.

- size'xattr:trusted.glusterfs.quota.size=0x000000000 00 00000000000000000000000000000000001. The 'size' xattr corresponds to the size of the directory, where the first sixteen digits refer to the actual size, the next sixteen bits refer to file count and the next sixteen digits refer to the directory. Size xattr is present only fordirectories.
- Dirty xattr - In backend dirty xattr always will be zero. Whenever we need to account, we have tomaintainconsistency and dirty xattr is used for it. For example: Consider we have : '/dir/file' In 'file', contri is updated, and before we update 'dir', some problem happens(may be a brick has crashed), and so 'dir' is not updated. Now we have 'dirty' set to 1(whenever a write or some operation that modifies file happens, dirty xattr will be set to 1) and then once 'contri' is updated to 'dir', we change the 'dirty' value back to 0. This way it prevents from misaccounting. During lookup, we inspect the 'dirty' flag if it is zero, else(which meansdirty is set to 1, probably an operation has happenedand not updated successfully), we check the sum of size of files under the directory, update the size and contri in the directory(from leaf to root) and make dirty again set to0.

The current mechanism stores count of objects/files as partof extended attribute of a directory. Each directory will havethe numberoffilespresentinatreewithtreebeingconsideredas the root of the directory (Applies to subdirectories as well). Hence, crawling till root is not needed to retrieve the number of files(objects). There is something called as timeoutswhich also helps in the enforcement. Let's look on what is called astimeouts.

B. Timeouts

Memory cache size needs to be updated in order to make the enforcer work properly. For performance reasons, quota caches the directory sizes on client. You can set timeout indicating the maximum valid duration of directory sizes in cache, from the time they are populated. For example: If there are multiple clients writing to a single directory, there are chances that some other client might write till the quota limit is exceeded. However, this new file-size may not get reflected in the client till size entry in cache has becomestale because of timeout. If writes happen on this client during this duration, they are allowed even though they would lead to exceeding of quota-limits, since size in cache is not in sync with the actual size. When timeout happens, the size in cache is updated from servers and will be in sync and no further writes will be allowed. A timeout of zero will force fetching of directory sizes from server for every operation that modifies file data and will effectively disables directory size caching on clientside.

C. Challenges and the EnforcerTechniques

When you enable quota, glusterdinitiates "find . | xargs stat", just to lookup on each file and create the xattr's. When you disable quota, we send "find . xargs stat", which crawls each file, and removes the xttr. Consider we have a quota enabled volume. When quota is disabled on this volume let's see what are the steps being carried out.

- CLIwillsendtherequesttoglusterd.
- Glusterd will initiate a backend process 'find / — xargssetxattr- x....'tocleanupallquotaxattrsinthebackend.
- CLIwaitsfortheresponsefromglusterd.
- Glusterd will continue to execute the operation that CLI initiatedevenwhenonekillsCLI.
- As mentioned above, glusterd does a lookup on each file and then remove the quota xattr associated with the objects,buttherecouldbeproblemunderthescenario:
- The clean-up process may have terminated without completely cleaning-up the quotaxattrs.
- One may try to enable quota before the clean-upprocess hascompleted.
- In simple when quota is enabled again, this canmess-up the markeraccounting.

To be precise, let us consider an example, Example: Consider, we have /dir/file where 'file' is of size 5MB, so '/dir' and '/' will be accounted as 5MB. Here the cleanup process of cleaning xattr of 'file' is done, but failed to cleanup xattr on'/dir'and'/'.Itmaybebecausetheprocesshasterminated or due to a failure in node. Now if quota is enabled, 'file' will be accounted 5MB, this 5MB is added to '/dir' and '/' but '/dir' and '/' already has 5MB accounted which was not cleaned during cleanup and after adding 5MB the quota usage becomes 10MB which makes markeraccounting messy.. Hence as a solution to the above problem, Quota versioning came intopicture.

D. QuotaVersioning

A version number is suffixed for all the quota xattrs. This version number is specific to marker xlator (i.e) when quota xattrs are requested by quotad/client, marker will remove the version number which is suffixed in the key before sending the response. Previously the xattr's were just

- 'trusted.glusterfs.quota.size'
- 'trusted.glusterfs.quota.limit-set'. Now we are

suffixing aversionnumberattheendwhichmakes,

- ’trusted.glusterfs.quota.size.<versionnumber>’
- ’trusted.glusterfs.quota.limit-set.<version number>’ where version number starts from 1 and keeps incrementing by 1 till N. The same concept applies to the ’contri’ xattr aswell.

With this approach, every time quota is enabled the xattr’s gets attached with a new number and is treated like a separatesession.Eventhoughtheproblemofmessingmarker accounting is fixed, the previous xattrs are just left without cleaning up, leaving extra space. This can be fixed by checking up the version number in xattrs when quota is enabled and if xattrs with older version number exists, just cleanup them and then create new xattrs. Let’s look on the projects being worked uponcurrently.

E. Techniques that Needs to be Enhanced

- PerformanceIssue-Currently,whenfilesaredistributed across bricks, just for a lookup, currently we are send- ing, from client->dht ->replica- >brick, instead wecan activate one process on one brick, which could increase theperformance.Thisworkisunderprogress.
- Recursive Directories Problem in Enforcer - For exam- ple: Consider the directory structure /1/2/3/4/5/6/7/f1. One has to keep checking till the root to checkthe limit and do the enforcement if it exceeds andperform

  accounting accordingly which could cause serious per- formance problem. In quota, we always need to crawl till the root to do the accurate enforcement. This needs to be fixed from the scratch.
- Issue in Enforcing - When a brick is down, then quotad, the client process do not get the aggregated value and it instead sends zero, and the write can happen more than the limit. Say, there are two bricks, b1 and b2, brick b1 has size 10GB written and the limit set on b1 is 15GB but b1 went down, so brick b2 thinks that 15GB is free in brick b1 and tries writing 15GB which could cause a problem inenforcement.

V. Conclusion

The Gluster file system is a revolutionary step forward in data management on every axis and in every dimension: absolute performance; scalability of performance and capac- ity; manageability at scale; ease of use; reduced cost of ac- quisition, 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. Also, depending on the need, the features are exported to the users.
Onesuchimportantfeatureasseeninthisarticle,isEnforcer. Enforcer finds its usage in quite a lot of applications.It majorly includes the cloud applications, banking sectors, educational universities, scientific researches and generically in places where sectors(or disk space) need to be allocated to theusers.

As a whole, 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. To completely precise the entire article, here is the bottom line, ”Gluster - data management for the 21st century, leaving the past behind and Enforcer, an important component in GlusterFS, which will never let you off limits andisextremelyusefulfortherealtimeapplications.”

References

[1] Gluster Whitepaper, 2010, ”GlusterFS Architecture”, [Online]. Available:http://download.gluster.com/pub/gluster/documentation/Gluster Architecture.pdf.[Accessed: 22-Dec-2015].

[2] Dan Lambright, SA Summit, 2014, ”Erasure Codes and Storage Tiers on Gluster”, [Online].Available: http://www.slideshare.net/Red Hat Storage/erasure-codes- and-storage-tiers-on-gluster. [Accessed: 19-Nov-2015].

[3] Gluster Community, 2015, ”GlusterFS Documentation”, [Online]. Available: https://gluster.readthedocs.org/en/latest/Quick-Start-Guide/Architecture/. [Accessed: 07-Oct-2015].

[4] Gluster Community, 2010, ”Cloud Storage for the Modern Data Center, An Introduction to Gluster Architecture”, [Online]. Available: http://moo.nac.uci.edu/ hjm/fs/An Introduction ToGluster ArchitectureV7 110708.pdf. [Accessed: 12-Oct-2015]

[5] Vijay Bellur, Vault Conference, 2015, ”An Introduction to GlusterFS”, [Online]. Available: https://lwn.net/Articles/637437/. [Accessed: 20-Jan-2016]

[6] ManikandanSelvaganesh, 2016, ”GlusterFS - Quota”, [Online]. Available:https://manikandanselvaganesh.wordpress.com/. [Accessed: 10-Feb-2016]