**Hengtian**

**Introduction to GlusterFS**

**The white paper**

**Combined by Ricky**

([xintan@hengtiansoft.com](mailto:xintan@hengtiansoft.com))

# Distributed File System of GlusterFS

### Introduction

GlusterFS is the core of the scale-out storage solution Gluster, it is an open source distributed file system, which is equipped with powerful scale out ability, and can support several PB size of disk space and thousands of clients (See ). GlusterFS gathers physically distributed storage resources to a unique, global namespace by TCP/IP or InfiniBand RDMA network. GlusterFS was designed based on scalable user space; it performs well on different data workload.

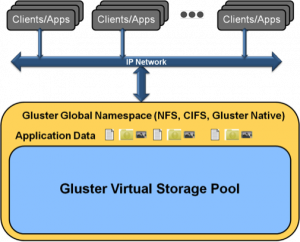
****

Figure the unified mount point of GlusterFS

GlusterFS can be accessed from all kinds of standard clients of standard applications run on whatever standard IP network, as the Figure 5 shows; users can access the application data by standard protocols such as NFS/CIFS in the unique global namespace. GlusterFS make it possible for users to get rid of original expensive storage system, and deploy virtualized storage pool which support centralized management and scale-out by cheap storage devices. The capacity can be scaled to TB/PB level. The main features of GlusterFS are as follows:

#### Scalability and High Performance

GlusterFS make full use of the two features: scale-out architecture and elastic hash, to provide a high scalability storage solution which can support several TB, even PB disk space.

Scale-out architecture allows users to increase the storage capacity and performance by simply add resources, disk, computing and I/O resources can be added independently.

Elastic Hash allows GlusterFS to get rid of the metadata server; it eliminates the single point of failure and performance bottleneck, and implements a real parallel data access.

#### High availability

GlusterFS can make a duplicate of the files automatically, such as mirror or multi-backup, to make sure that the files are always accessible, even if a hardware breakdown occurred. The self-healing function could restore the broken data to normal state, and what’s more, the restore process runs at backend incrementally, so it won’t make any performance problem to the system. GlusterFS didn’t use a specified disk file system designed for itself, instead, it uses the existing standard file system already installed on the operating system (EXT3, ZFS, etc.), yet all standard tools can be used to copy and access files.

#### Unified Global Namespace

All disk and memory resources are able to be integrated to one single virtualized resource pool by this unified global namespace, and it hides all details of the low-level physical hardware from upper level users. Storage resources can be scaled elastically in resource pool according to demands, for example, shrinking or enlarging.

#### Elastic Hash Algorithm

GlusterFS use the algorithm named elastic hash to locate the date in storage pool, not centralized or distributed metadata server index. On other scale-out storage system, the metadata server always courses a bottleneck on I/O performance and single point of failure. All storage nodes in Gluster could locate all data piece intelligently, and no longer need to check the index or send a query to other server. This mechanism makes the data access parallel, and implements a real linear scaling.

#### Elastic Volume Management

Data are stored on logic volumes, volumes are partitioned independently from the virtualized physical storage pool. Storage server can be added and removed on the run, without disrupting the system. All these operations like shrinking or enlarging a logic volume on any of the configuration server, moving a volume from one to another, adding or removing a sub storage system, are able to be flexibly adapted without disrupting any running functionality.

#### Based On Standard Protocol

Gluster server support NFS, CIFS, HTTP, FTP and gluster original protocol, and it is fully compatible with POSIX standard. Existing application can access the data in Gluster without any modification, or by using the specified API. This is very useful when deploy gluster on public cloud, gluster will give an abstraction to the API of cloud server provider, and provide a standard interface of POSIX.

### Gluster Design Goals

The design philosophy of GlusterFS is quite different from other existing parallel/cluster/distribute file system; for it is very hard to compete with Lustre, PVFS2, Ceph, etc. without an essential breakthrough on design, not even mention some commercial file system which had been proofed good after years of testing and improving both on technique and market such as GPFS, StorNext, ISILON, IBRIX, etc. Gluster was designed to achieve several major goals:

#### 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. On early version of GlusterFS, it was necessary to stop the services for some kinds of maintenance, this has been improved after version 3.1.X. GlusterFS achieves this goal mainly by virtualization technology and logic volume management.

#### Linear Scaling

Usually speaking, linear scaling is hard to implement, in fact, the 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. Nowadays many system has a incredible scalability, for example, Luster has an ability to support over 1000 storage node and over 25000 clients, but it isn’t linear scaling.



Figure Linear VS Logarithmic Scaling

Scale-up is such a notion that to increase the storage capacity or performance on a single node, while scale-out is to add storage nodes, so that the entire system will get better. But often, scale-up can’t fulfill the requirement of storage according to physical and theoretical limits. So scale-out has become a popular technology.



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. 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. GlusterFS gain 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



Figure Linear Scaling in Gluster

#### High Reliability

Similar to GFS (Google File System), GlusterFS was constructed on common server and storage devices, so the reliability is particularly critical. At the beginning of the design, GlusterFS takes reliability as the core factor that to be considered, and adopts many technologies to achieve this goal. First, Gluster considers a hardware failure as normal event, e.g. disk or network breakdown, even a misoperation by administrator, which may course data damage. GlusterFS was designed to support auto mirror and self-healing to provide high reliability without management from the administrator. Second, GlusterFS takes the advantage of the EXT3/ZFS file system’s log component to make the data more reliable. Third, GlusterFS has no metadata server, there’s no need to synchronize or consistent, so it simplified the system complexity. Only raised the performance, but also increased reliability.

### Technical Characteristics

There are seven fundamental technical differentiators between Gluster and traditional systems, which are listed below:

#### Software Only

Gluster believes that storage is a software problem - one which cannot be solved by locking customers into a particular vendor or a particular hardware configuration. Gluster was designed 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.

#### Open Source

Gluster believes 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 <http://www.gluster.org>.

#### Complete Storage Operating System Stack

Gluster believe 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.

#### 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.

#### Modular, Stackable Architecture

Gluster is designed using a modular and stackable architecture (Figure 4). To configure Gluster for highly specialized environments (e.g. large number of large flies, 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 Modular and Stackable Architecture

#### 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.

#### 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.

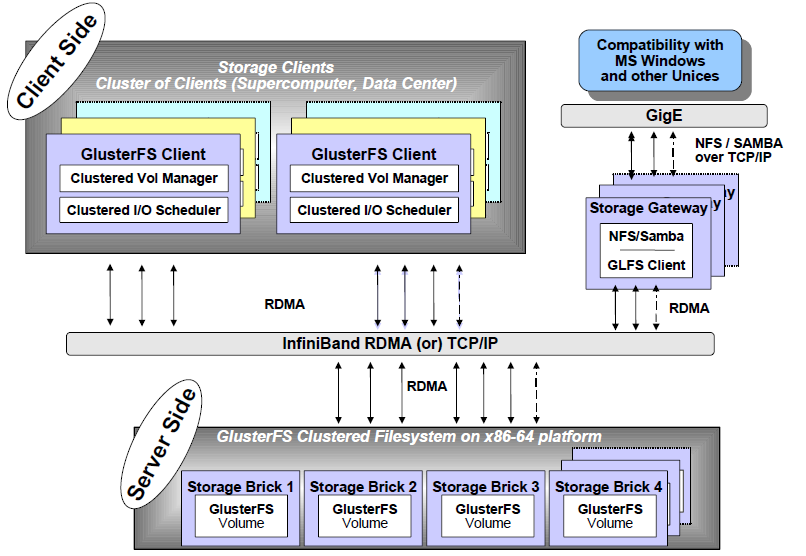


Figure 5 architecture and organization of GlusterFS

### Overall Architecture and Design

The overall architecture is showed in , it consists of brick server, client and NFS/Samba gateway. It isn’t hard to notice that gluster has no metadata server, which is a main feature of gluster, this will give a great help on promoting the performance, reliability and stability of the entire system. TCP/IP and InfiniBand RDMA are well accepted in gluster, and gluster client could access data through original Glusterfs protocol, other terminal without gluster client may use NFS/CIFS, too.

Brick server mainly provides disk space for the system, the file data are distributed to different brick servers by some kind of scheduling strategy. There’s a daemon process runs on it, handling the data requests from other components. The data are stored as normal files on its local file system, such as EXT3, EXT4, XFS, ZFS, etc. Multi server may be combined to build much more complicated clusters, e.g. Stripe (RAID0), Replicate (RAID1), DHT (Distribute Hash), and of course, RAID 10, etc., via volume configuration of clients or storage gateway.

Because of the elimination of metadata server, client has to do more things, including volume management, I/O scheduling, file locating, data caching, and so on. The client runs a daemon named Glusterfs, which in fact is a symbolical link to Glusterfsd, to mount GlusterFS on local file system with the help of FUSE (File System in User Space), so it is well compatible with POSIX standard. After version 3.1.X, client needn’t maintain the volume information independently, this job was simplified by moving it to a elastic volume management server—glusterd which runs on gateway. The workload of GlusterFS client is a little bit higher than traditional distribute file system, including CPU occupancy rate and memory use.

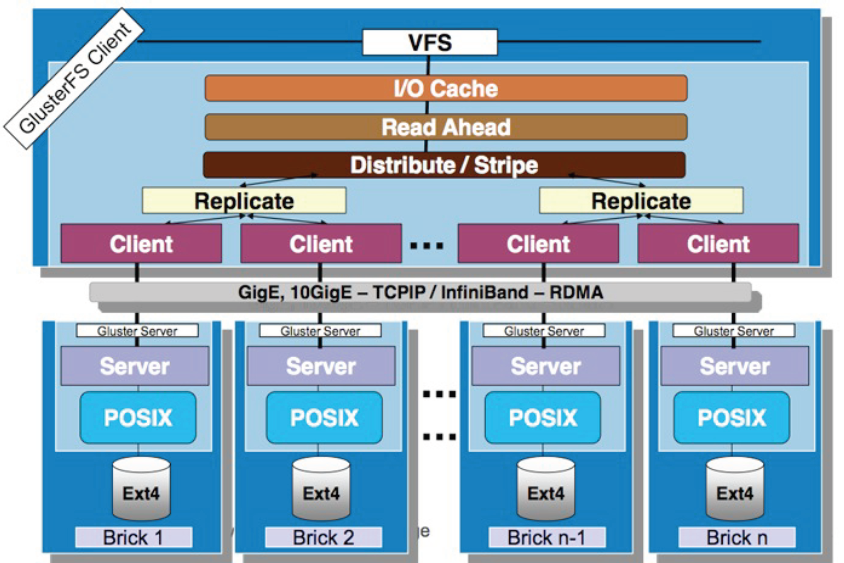
****

Figure the modular stack design of GlusterFS

GlusterFS implements a architecture of modular stack, as is showed in . A module is called Translator, which is a powerful mechanism provided by GlusterFS and may support efficient and simple extension. Every translator is a dynamic library, and each functionality is implemented by it, e.g. Cluster, Storage, Performance, Protocol, Features, etc.

### Translators

As has mentioned before, Translator is a powerful functionality extension mechanism provided by Gluster. This idea was referenced from GNU/Hurd Micro Kernel operating system. In GlusterFS, there’re such kinds of translators:

（1）  Cluster：Distributed storage cluster, it has three types till now, AFR, DHT and Stripe.

（2）  Debug：Tracking the internal function call and system call.

（3）  Encryption：Implements a simple data encryption.

（4）  Features：Access control, lock, Mac compatibility, read only, quota, etc.

（5）  Mgmt：Elastic volume management

（6）  Mount：implementing of FUSE interface.

（7）  Nfs：Internal NFS server

（8）  Performance：Performance optimization of I/O cache, threads, quick-read, read-ahead, stat-prefetch, sysmlink-cache, write-behind, etc.

（9）  Protocol：Implement of server and client protocol

（10）Storage：Implement of POSIX interface

It is necessary to introduce Cluster Translator. It is the core to implement GlusterFS, and consists of AFR (Automatic File Replication), DHT (Distributed Hash Table) and Stripe.

AFR likes RAID1, it will make multi copy on different storage node, for the use of self-healing and reliability. AFR will dispatch the request to all storage nodes while reading for a better performance. And it locks the file on all lock-servers while writing; it treats the first node as the lock-server by default, and can be specified one or more. After a successfully writing on all nodes, it unlocks the file as commonly do. AFR will check the data consistency and restore them. The self-healing progress will be trigged when it comes the first visit on the file directory.

DHT is the so called elastic hash algorithm, it distribute files by hashing. It doesn’t rely to the namespace while finding a file, just simply apply the hash algorithm. But when it comes to traverse a directory, the implement is complicated and the performance is low, because it has to search on all nodes. One single file will be scheduled to a single node, after the file is located, the read/write mode is quite simple. DHT has no fault-tolerant, usually we use it together with AFR, as .

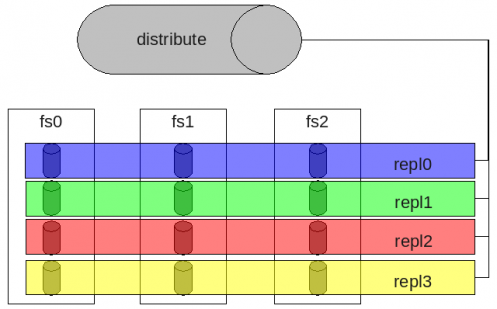
****

Figure DHT+AFR

Stripe like RAID0, a file is divided to fixed length pieces and stored on all nodes applying the Round-Robin algorithm. All nodes of Stripe make a complete namespace, so the reading and writing operation could be executed parallel on multi-nodes, which shows a wonderful performance! Of course, Stripe is also not fault-tolerant.

### Setup

#### Prepare

GlusterFS is an open source distribute file system which has some very good features, here we prepare real machines to setup a small distribute file system with only two servers.

|  |  |
| --- | --- |
| **Hardware** | **Description** |
| CPU | Pentium ® Dual-Core E5300 @ 2.60GHz |
| RAM | 1 GB |
| Hard Disk | 66 GB |
| Network Card | 100baseTx-FD |
| Switch | D-Link 10/100 Fast Ethernet Switch |

|  |  |
| --- | --- |
| **Software** | **Description** |
| Operating System | Ubuntu 10.04 server |
| Kernel | Linux Ubuntu 2.6.32-28-server |

Connect these two servers to the same trusted sub-network.

|  |  |
| --- | --- |
| **Server** | **IP** |
| Server1(also as client) | 172.17.13.28 (whatever as you like) |
| Server2 | 172.17.13.29 |

Now, start the server, and make sure that the networking connection works fine. And, also, we have to install some software from the mirror, so please make sure all the servers can access the Internet freely.

#### Software Install

Now, we begin to install some basic software to support Gluster.

* 1. Install from DEB

We can download the DEB file from: <http://download.gluster.com/pub/gluster/glusterfs/3.1/LATEST/Ubuntu/>

Then we get glusterfs\_3.1.7-1\_amd64.deb, type the command below in Terminal (Need root privilege):

cd /path/to/the/file

dpkg –install glusterfs\_3.1.7-1\_amd64.deb

* 1. Install from mirror

We can also install GlusterFS from mirror. First, add sources which you can access from the server to the APT source list, then run the following command:

apt-get update

apt-get install gluster-server gluster-client

*Note:* Only one of the servers needs to install gluster-client, here is Server1.

#### Configuration

We’ll see that we have some new command now if we type in “glu” then press TAB button.

glusterfs glusterfsd

Before we start up gluster client, we have to configure these two gluster file server first.

* 1. Server Configuration

Type the commands below to make 4 directories, which will be used as storage volume.

root@ubuntu:~# cd /root/

root@ubuntu:~# mkdir -p gluster/bricks{0,1,2,3}

root@ubuntu:~# tree

.

└── gluster

├── bricks0

├── bricks1

├── bricks2

└── bricks3

5 directories, 0 files

Go to ***/etc/glusterfs***, we can see two files here: **glusterfs.vol, glusterfsd.vol**.

root@ubuntu:~# cd /etc/glusterfs/

root@ubuntu:/etc/glusterfs# ls

glusterfsd.vol glusterfs.vol

For server, we only have to open **glusterfsd.vol**

root@ubuntu:/etc/glusterfs# vim glusterfsd.vol

edit the content as follows(On ***Server1***):

### file: client-volume.vol.sample

#####################################

### GlusterFS Client Volume File ##

#####################################

#### CONFIG FILE RULES:

### "#" is comment character.

volume brick0

type storage/posix # POSIX FS translator

option directory /root/gluster/bricks0 # Export this directory

end-volume

volume brick1

type storage/posix # POSIX FS translator

option directory /root/gluster/bricks1 # Export this directory

end-volume

volume brick2

type storage/posix # POSIX FS translator

option directory /root/gluster/bricks2 # Export this directory

end-volume

volume brick3

type storage/posix # POSIX FS translator

option directory /root/gluster/bricks3 # Export this directory

end-volume

### Add network serving capability to above brick.

volume server

type protocol/server

option transport-type tcp

# option client-volume-filename /etc/glusterfs/glusterfs-client.vol

subvolumes brick0 brick1 brick2 brick3

# NOTE: Access to any volume through protocol/server is denied by

# default. You need to explicitly grant access through # "auth"

# option.

option auth.addr.brick0.allow 127.0.0.1 # Allow access to "brick" volume

option auth.addr.brick1.allow 127.0.0.1 # Allow access to "brick" volume

option auth.addr.brick2.allow 127.0.0.1 # Allow access to "brick" volume

option auth.addr.brick3.allow 127.0.0.1 # Allow access to "brick" volume

end-volume

And on ***Server2,*** do the same thing but the last few lines have to be modified to:

volume server

type protocol/server

option transport-type tcp

# option client-volume-filename /etc/glusterfs/glusterfs-client.vol

subvolumes brick0 brick1 brick2 brick3

# NOTE: Access to any volume through protocol/server is denied by

# default. You need to explicitly grant access through # "auth"

# option.

option auth.addr.brick0.allow 172.17.13.28 # Allow access to "brick0" volume

option auth.addr.brick1.allow 172.17.13.28 # Allow access to "brick1" volume

option auth.addr.brick2.allow 172.17.13.28 # Allow access to "brick2" volume

option auth.addr.brick3.allow 172.17.13.28 # Allow access to "brick3" volume

end-volume

*Note:* The server only provides disk space to store the files, of course we can let one server has only one storage volume, and configure 8 or more servers. Here is just an example.

* 1. Client Configuration

Because GlusterFS requires FUSE module, so first make sure that the FUSE module is installed properly in your Linux kernel. (More information about FUSE, please visit <http://fuse.sourceforge.net/> )

Change to directory ***/etc/glusterfs***, backup the original file **glusterfs.vol** to **glusterfs.vol.bg** , and clear the content.

root@ubuntu:~# cd /etc/glusterfs/

root@ubuntu:/etc/glusterfs# mv glusterfs.vol glusterfs.vol.bg

root@ubuntu:/etc/glusterfs# cat /dev/null > glusterfs.vol

Then add the content below:

### Add client feature and attach to remote subvolume

volume client0-0

type protocol/client

option transport-type tcp

option remote-host 127.0.0.1 # IP address of the remote brick

option remote-subvolume brick0 # name of the remote volume

end-volume

volume client0-1

type protocol/client

option transport-type tcp

option remote-host 172.17.13.29 # IP address of the remote brick

option remote-subvolume brick0 # name of the remote volume

end-volume

volume bricks0

type cluster/replicate # replicate means the two volume

# will write synchronously

subvolumes client0-0 client0-1

end-volume

volume client1-0

type protocol/client

option transport-type tcp

option remote-host 172.17.13.29 # IP address of the remote brick

option remote-subvolume brick1 # name of the remote volume

end-volume

volume client1-1

type protocol/client

option transport-type tcp

option remote-host 172.17.13.29 # IP address of the remote brick

option remote-subvolume brick1 # name of the remote volume

end-volume

volume bricks1

type cluster/replicate

subvolumes client1-0 client1-1

end-volume

volume filesys

type cluster/distribute # distribute means the two volume

# is treated as one volume, it likes

# RAID0, but not totally. One of

# the volume will be chose while

# writing files somehow.

subvolumes bricks0 bricks1

end-volume

### Add readahead feature

#volume readahead

# type performance/read-ahead

# option page-size 1MB # unit in bytes

# option page-count 2 # cache per file = (page-count x page-size)

# subvolumes client

#end-volume

#### Testing

Now everything is ready, let’s start the services! On ***Server1*** and ***Server2***, type in:

root@ubuntu:~# glusterfsd

and open the log file at ***/var/log/glusterfs/-etc-glusterfs-glustefsd.vol.log***.

root@ubuntu:~# tail /var/log/glusterfs/-etc-glusterfs-glustefsd.vol.log

If you see **“Successfully started”**, then congratulations! If not, according to the logs and figure out what’s wrong.

After both of the servers have successfully started, we can use their disk space by mount it to the client’s local file system:

root@ubuntu:~# mount –t glusterfs /etc/glustefs/glustefs.vol /mnt

root@ubuntu:~# cd /mnt

root@ubuntu:/mnt# df -h

Filesystem Size Used Avail Use% Mounted on

/dev/mapper/ubantu-root

71G 1.1G 66G 2% /

none 490M 204K 490M 1% /dev

none 495M 0 495M 0% /dev/shm

none 495M 60K 495M 1% /var/run

none 495M 0 495M 0% /var/lock

none 495M 0 495M 0% /lib/init/rw

/dev/sda1 228M 17M 199M 8% /boot

/etc/glusterfs/glusterfs.vol

7.5G 3.2G 4.0G 45% /mnt

Let’s make some test by using these commands:

root@ubuntu:/mnt# touch test\_file{0,1,2,3}

root@ubuntu:/mnt# mkdir test\_dir{0,1}

root@ubuntu:/mnt# ls

test\_file0 test\_file1 test\_file2 test\_file3 test\_dir0 test\_dir1

And on Server1:

root@ubuntu:~# tree

.

└── gluster

├── bricks0

│   ├── test\_file0

│   ├── test\_file2

│   ├── test\_dir0

│   └── test\_dir1

├── bricks1

│   ├── test\_file1

│   ├── test\_file3

│   ├── test\_dir0

│   └── test\_dir1

├── bricks2

└── bricks3

On Server2:

root@ubuntu:~# tree

.

└── gluster

├── bricks0

│   ├── test\_file0

│   ├── test\_file2

│   ├── test\_dir0

│   └── test\_dir1

├── bricks1

│   ├── test\_file1

│   ├── test\_file3

│   ├── test\_dir0

│   └── test\_dir1

├── bricks2

└── bricks3

We see that Server1 and Server2 have the same content, and if we delete a file from one Server1, it will recover after some time automatically!

The architecture is as follows:

Brick0

Brick0

replicate

replicate

Brick1

Brick1

replicate

replicate

Brick2

Brick2

replicate

BrickN

BrickN

replicate

……

……

**GlusterFS**