前言

为什么写这本书?

对于这个问题，萌生这种想法是从 2020 年下半年，因为转变方向开始接触 glusterfs 开始的。对于 glusterfs，一开始在网上搜了很多资料，但是大部分资料的内容都是残缺不全的，并且没有一本专业系统的技术书来专门讲述 glusterfs。而 glusterfs 作为世界上与 ceph、hdfs 齐名的分布式存储系统，但是资料却完全无法和这两者相比，这是不应该的。因此在 2021 年五月下旬的时候，和朋友一起聊天时，激起了想专门写一本 glusterfs 书的想法，抱着尝试的心态，通过阅读源码资料，还有国内外的一些信息，慢慢不断地整合学习，才有了这本书的诞生。当然这本书作为本人工作生涯的第一本书，写下这本书的时候，恰好也是工作第三年了，对于程序员的职业生涯来说，这算是迈向了一个新的阶段了，给自己一个新的礼物吧。

读者对象

这本书因为是希望对 glusterfs 有一个相对全面系统的认识，因此在部分章节会有大量的源码阅读内容，而这本书内容，对于理解每一个系统工具都是必经之路，不同基础的读者阅读时可能会感受不一样，如果在阅读过程中，遇到部分细节没有弄清楚或者一时半会不知道的时候，建议先宏观了解每个章节的宏观思想，然后后面有恰当的机会再回头重新阅读，那样或许会有新的感受和发现。

另外对于本书的读者对象，本书适合以下人群:

1. 想要深入了解 glusterfs 的，但是之前只是大概理解一些运维操作，对原理不熟悉的。
2. 有一定 glusterfs 运维基础经验，并且对 Linux 系统有使用经验，对 VFS 这些文件系统概念有简单认识的人。
3. 对分布式系统感兴趣的互联网工程师和架构师。

如何阅读这本书

本书主要分为五大部分：

简单使用篇(第一章): 这一部分的内容，主要是讲解在部署好了 glusterfs 系统之后，如何简单地创建并且使用不同类型的 volume,还有讲解了一下仲裁节点的特点，和集群维护中的日志文件位置等。

基础概念篇(第二章): 这一部分的内容主要是讲解一下 Linux 文件系统的一些基础概念基础，对 VFS 的介绍等，为后面的章节部分内容做好一些基础概念的准备。

原理篇(第三章): 这一部分的内容，主要就是讲解了 glusterfs 的核心概念，包括 gfid 和 posix 接口，线程模型等，这部分内容主要以源码分析为主，阅读难度较大，对于具体细节的内容如暂时无法理解的，建议先宏观了解每个章节的思想和内容，后续再考虑深入每个章节的部分内容。

性能特性篇(第四和五章): 这一部分的内容，主要讲解 glusterfs 的特性和性能参数相关的，包括扩缩容和容量限制等。

运维篇(第六章): 这一章主要分享了在日常运维使用过程中遇到的生产环境版本的 bug 问题，还有目前和未来 glusterfs 的一些优化内容与项目发展计划等。

勘误和支持

由于本人水平有限，写这本书时是本人工作第三年同时也是第一次写书，因此如有错误和不当之处，尽请指出，或者阅读时有疑惑不解时，可以联系本人联系，github 地址为: github.com/httaotao,谢谢。

目录

1.第一章让集群先跑起来

1.1. 我们为什么要用 glusterfs

在这里我想简单介绍一下，什么是 glusterfs,在官方的介绍里面,这里提到 GlusterFS 是可扩展的网络文件系统，适用于数据密集型任务，例如云存储和媒体流。以上这段话是从 glusterfs 官方网站[1] 上面获取的信息。 而目前对于 glusterfs 而言，个人觉得最大的特点是无中心架构的分布式文件系统.其中和我们熟悉的 master/slave 架构的系统有着非常不同的地方。而我们这里将要介绍的，也是基于这个特点下，去逐渐了解 glusterfs 的一些特性与使用方式。

在了解一下新的工具，尤其是分布式系统的时候，总会有一些相似的问题，为什么我们要使用这个工具？glusterfs 能够带给我们什么？如果你还没有真正投入生产环境使用，在做调研测试阶段的话，那么我们又该如何去熟悉一些新的文件系统呢？关于这个问题，我希望在最后的时刻才回答，因为当我们真正去深入了一些不同的系统模块之后，未来在面对新的系统架构工具，我们才会有一些明确地目标和方式去进行对比。 同时基于这些内容，我们才会有更多的思考与抉择。

相反，如果在内容的最开始就讲解这些问题，那么可能大家会比较迷茫与无法理解，或者说没有一个明确的印象。当然如果你已经在生产环境中有了比较多的 glusterfs 的使用经验，那么也可以直接去阅读感兴趣的内容，而不需从部署与一些特性开始关注。

这里先约定一下,后面所有的讲解内容，如果没有特别指明，那么默认就是本人写该资料时官方最新的版本，也就是 9.2。同时关于实验环境，这里建议至少创建四台台虚拟机进行测试使用，三台虚拟机是一个小集群，然后一台客户端机器。在部分场景下，可能需要更多的虚拟机，操作系统版本等则没有严格要求，而本人的测试环境测试 centos7 的系统，使用 ubuntu 或者其他 linux 系统皆可。

另外这里为了保持一下风格，或者说不显得那么怪异，有一些专业名词将保持使用英文名词，这样可能在未来大家接触官方的英文文档，或者在 github 上面提 issue 的时候，并不会感到比较陌生和困惑，同时可能未来也会有更多关于 glusterfs 的书籍，那么阅读起来的话，会更有统一性和流畅性。

1.2. 先让 glusterfs 跑起来

1.2.1. 简单概念介绍

在讲解 glusterfs 的 volume 和 brick 之前，这里为了方便大家理解，需要先简单介绍一些概念，主要是关于 glusterfs 中的常见的。当然，有一些概念可能会放到后面讲到的时候再介绍，避免一下子讲解太多陌生概念导致的困惑与迷茫。

这里 glusterfs 的 brick 可以简单理解为数据存储在节点上面的目录及其管理的进程，而卷，也就是 volume 就是一份完整的数据存储在不同的 brick 上面的集合，这里会有关于设置这些存储时候的一些性能参数或者其他指标等。如果这段话不太理解也没关系，相信在后面创建 volume 之后，大家会有较为直观的感受与理解。

glusterfs 目前官方支持五种不同类型的卷。其中生产环境里面用的比较多的是分布式复制卷和分布式冗余卷. 下面将会简单介绍一下不同类型的卷的使用特点与方式。当然在讲解 volume 之前，希望分享一下关于 glusterfs 的一个特点，通信的全互联架构，因为 glusterfs 是没有主节点，也就是通常大家常见的 master 节点那样的概念，因此每一个客户端都会和不同的节点进行通信，具体的可以看看下图所示。

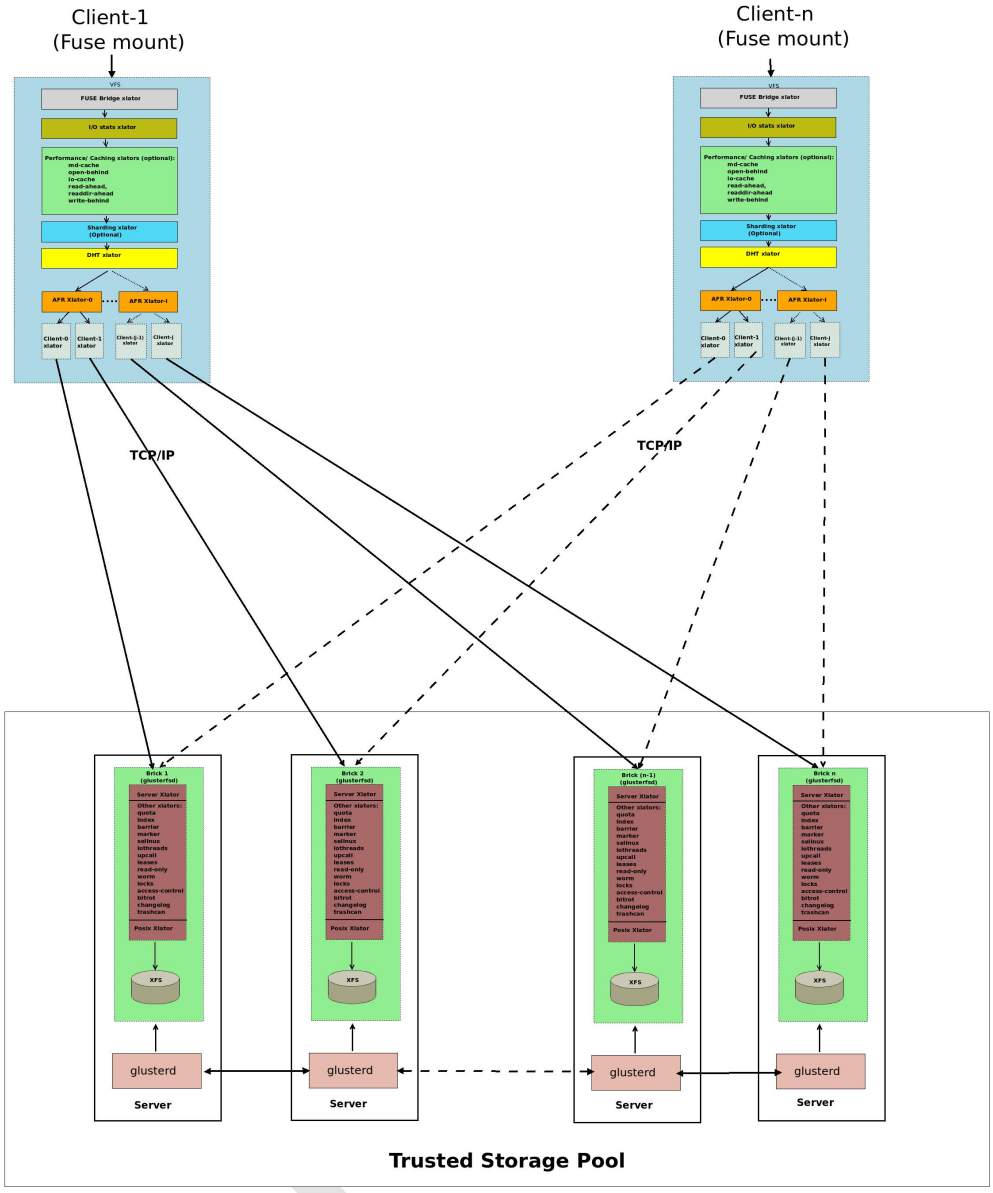


图 1.2.1-1　glusterfs 的客户端与服务端通信模型图

1.2.2.　一些有趣的 volume

1.2.2.1.　复制卷

官方支持复制卷和分布式复制卷，而生产环境中常用的是分布式复制卷，首先这里引用一些官方的图片，来理解一

下什么是分布式复制卷。

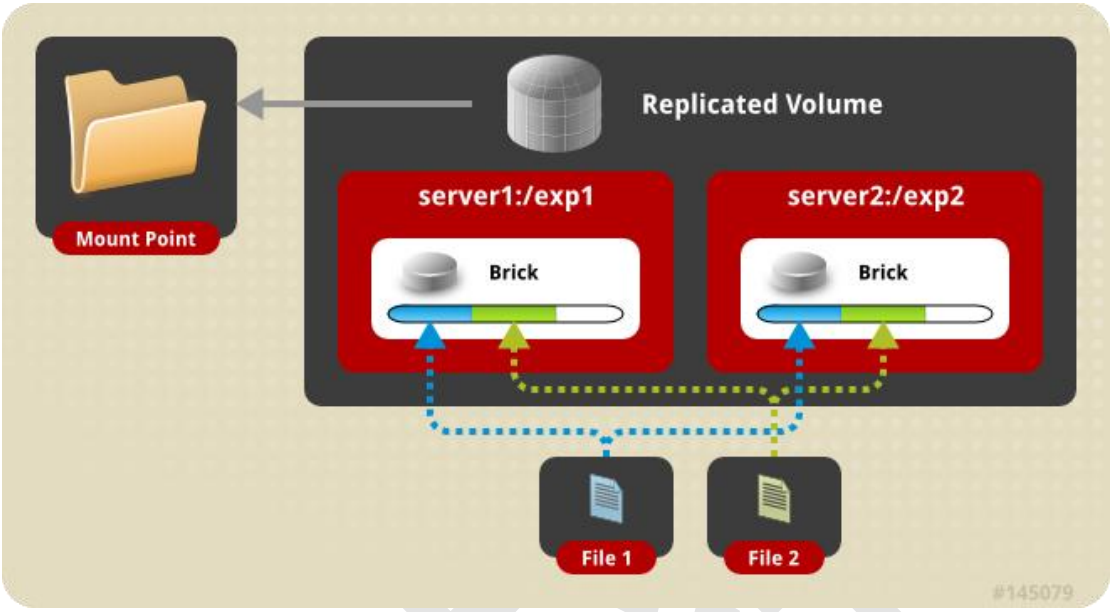


图 1.1

这里图片给出的是一个 2 个副本的复制卷，生产环境中通常使用 3 个副本的复制卷，创建的命令如下所示.

```
[root@gfs01 ~]#  gluster volume create test-replica replica 3 192.168.0.{110,111,112}:/glusterfs/test-replica
volume create: test-replica: failed: The brick 192.168.0.110:/glusterfs/test-replica is being created in the root partition. It is recommended that you don't use the system's root partition for storage backend. Or use 'force' at the end of the command if you want to override this behavior.
[root@gfs01 ~]#  gluster volume create test-replica replica 3 192.168.0.{110,111,112}:/glusterfs/test-replica force
volume create: test-replica: success: please start the volume to access data
[root@gfs01 ~]# gluster volume start test-replica
volume start: test-replica: success
```

这里因为 glusterfs 官方推荐不使用 root 进行操作的，或者使用 force 命令强制创建，那么这里作为测试使用可以不用考虑，但生产环境安装部署的时候需要注意该问题。这里使用命令创建了一个名为 test-replica 的三副本的分布式复制卷，那么这里创建之后并且启动了。那么该如何知道该卷的状态是否正常呢？可以查看状态和挂载使用。

```
# gluster volume status   test-replica
Status of volume: test-replica
Gluster process                                 TCP Port  RDMA Port  Online  Pid
------------------------------------------------------------------------------
Brick 192.168.0.110:/glusterfs/test-replica 49153     0          Y       16485
Brick 192.168.0.111:/glusterfs/test-replica 49152     0          Y       16368
Brick 192.168.0.112:/glusterfs/test-replica 49153     0          Y       16422
Self-heal Daemon on localhost               N/A       N/A        Y       16502
Self-heal Daemon on gfs03                   N/A       N/A        Y       16439
Self-heal Daemon on gfs02                   N/A       N/A        Y       16385

```

```
Task Status of Volume test-replica
------------------------------------------------------------------------------
There are no active volume tasks
```

这里重点需要关注两个地方，分别是 online 和 pid，这里如果每个节点上关于该 volume 的存储目录管理进程异常，也就是 brick 异常的话，那么这里就无法获取到对应的端口号的。那么这里还可以使用命令查看一下每个节点的 brick 进程，结果如下所示。

```
# ps -ef |grep 16485
root     16485     1  0 12:27 ?        00:00:00 /usr/sbin/glusterfsd -s 192.168.0.110 --volfile-id test-replica.192.168.0.110.glusterfs-test-replica -p /var/run/gluster/vols/test-replica/192.168.0.110-glusterfs-test-replica.pid -S /var/run/gluster/d403ef3161bd809c.socket --brick-name /glusterfs/test-replica -l /var/log/glusterfs/bricks/glusterfs-test-replica.log --xlator-option *-posix.glusterd-uuid=99827066-72c7-484f-b1e8-0a9c4bc46b54 --process-name brick --brick-port 49153 --xlator-option test-replica-server.listen-port=49153
```

这一段信息中，指定的日志路径默认都是在/var/log/glusterfs 下的，而 brick 的日志则在 /var/log/glusterfs/bricks 中，同时如果生产环境中遇到某个 brick 的进程异常，遇到需要手动启动 brick 进程的时候，可以使用上面的输出结果来手动执行，当然一般也很少会遇到这样的情况。

这里想先强调一下，对于复制卷，不建议使用 2 副本的复制，因为这样会在 brick 异常之后,比较容易出现脑裂的现象，关于这部分的内容，会在后面详细讲解一下。

那么这里该使用呢?可以在其他节点上,只要安装了 glusterfs client 的话，那么直接使用 mount 挂载即可。

```
[root@gfs03 ~]# gluster volume info test-replica

Volume Name: test-replica
Type: Replicate
Volume ID: 66fed900-29f5-40f5-8add-7e4e365ab29a
Status: Started
Snapshot Count: 0
Number of Bricks: 1 x 3 = 3
Transport-type: tcp
Bricks:
Brick1: 192.168.0.110:/glusterfs/test-replica
Brick2: 192.168.0.111:/glusterfs/test-replica
Brick3: 192.168.0.112:/glusterfs/test-replica
Options Reconfigured:
performance.client-io-threads: off
nfs.disable: on
transport.address-family: inet
storage.fips-mode-rchecksum: on
cluster.granular-entry-heal: on

```

```
[root@gfs03 ~]# mount -t glusterfs 192.168.0.110:test-replica /mnt/test-replica
[root@gfs03 ~]# ls /mnt/test-replica/
[root@gfs03 ~]# touch /mnt/test-replica/a.txt
[root@gfs03 ~]# date >> /mnt/test-replica/a.txt
...
[root@gfs03 ~]# md5sum  /mnt/test-replica/a.txt
9eb6c6683f293a1f62d38a4ed94b17c8  /mnt/test-replica/a.txt
[root@gfs03 ~]# md5sum /glusterfs/test-replica/a.txt
9eb6c6683f293a1f62d38a4ed94b17c8  /glusterfs/test-replica/a.txt


[root@gfs02 ~]#  md5sum /glusterfs/test-replica/a.txt
9eb6c6683f293a1f62d38a4ed94b17c8  /glusterfs/test-replica/a.txt

[root@gfs01 ~]#  md5sum /glusterfs/test-replica/a.txt
9eb6c6683f293a1f62d38a4ed94b17c8  /glusterfs/test-replica/a.txt
```

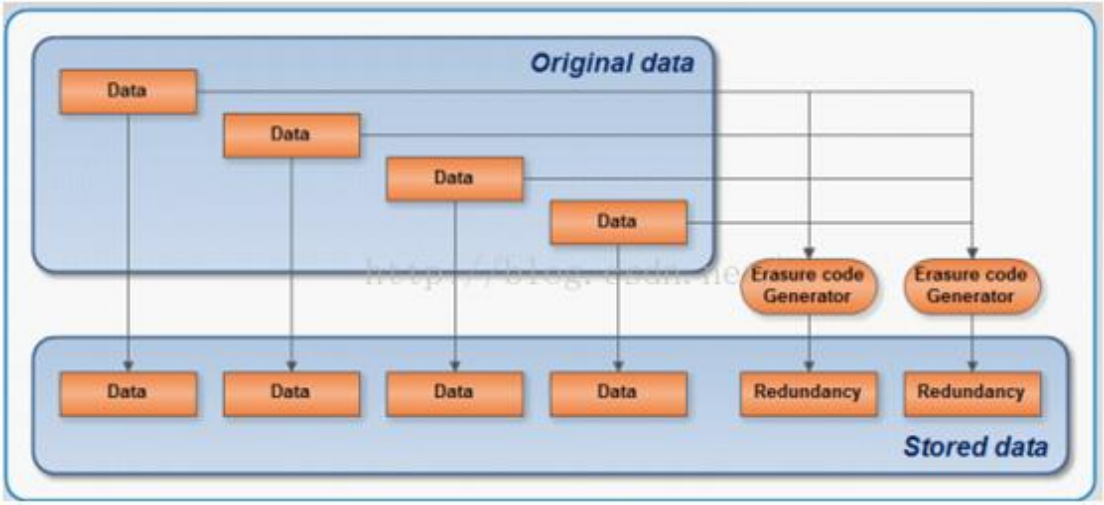
这里选择的 ip 节点可以是集群中的任意一个节点，哪怕 brick 并不在该节点上面，而这也正是 glusterfs 无中心架构的一个特点.因为通信方式采用的是全互联的模式,因此元数据信息保持一致的情况下，那么这里是可以选择任意节点进行挂载的。另外这里可以看到，复制卷的特点就是数据都是完整的一份的，然而冗余卷则是不一样的，可以进行对比一下.

当然在生产环境中，也不建议把挂载的 ip 都集中在其中的一个节点上， 否则所有的请求流量都会经过该节点，会加重节点的负载。而挂载以后，这里的使用就和本地目录类似了，也和 NFS 的使用类似。最后如果想解除挂载，那么直接使用 umount 目录即可。

```
[root@gfs03 ~]# umount  /mnt/test-replica/
```

1.2.2.2. 冗余卷

什么是冗余卷，这里与复制卷不一样的地方就是，数据到底以怎样的形式存放在节点上，像常见的 hdfs 是 3 副本机制的。而随着数据量越来越多，如果未来对于所有的数据都是 3 副本，那么占用的空间就会比较多了，因此为了节约空间，就有了冗余码的出现，通过下面的图来感受一下。



从图中可以很直观地感受到，所谓的冗余码，就是把一个完整的数据切割成多份，然后每一份计算得到一部分冗余码，并且存储在节点中，这里的存储空间会比完成的一份数据多一些，但是会远远比 3 副本所占用的空间小。同时因为有了冗余码，因此即使在损坏一定的比例数据下，数据的完整性都能得到保证。

那么这里的数据是如何存储地，冗余码如何计算得到的？通过计算得到这些冗余码等问题，这些将放在后面详细讲解，因为这部分内容涉及到比较多的数学内容，如果感兴趣，也可以先自行简单了解一下里德-所罗门码(Reed-solomon codes)。同时目前工业界也比较关注，未来到底是复制卷还是冗余卷，该如何使用，优缺点如何，都是非常值得思考与关注的问题。

这里我们先了解一下如何创建分布式冗余卷，并且这里我们挂载创建一个文件，写入一些数据，查看一下到底分布式冗余卷的数据是如何分布的。

```
[root@gfs01 ~]# gluster volume create test-disperse disperse 3 192.168.0.{110,111,112}:/glusterfs/test-disperse  force
volume create: test-disperse: success: please start the volume to access data

[root@gfs01 ~]# gluster volume start  test-disperse
volume start: test-disperse: success

[root@gfs01 ~]# gluster volume info test-disperse

Volume Name: test-disperse
Type: Disperse
Volume ID: 0390d729-b6d8-4edd-bb72-bd28c3ec7472
Status: Started
Snapshot Count: 0
Number of Bricks: 1 x (2 + 1) = 3
Transport-type: tcp
Bricks:
Brick1: 192.168.0.110:/glusterfs/test-disperse
```

```
Brick2: 192.168.0.111:/glusterfs/test-disperse
Brick3: 192.168.0.112:/glusterfs/test-disperse
Options Reconfigured:
storage.fips-mode-rchecksum: on
transport.address-family: inet
nfs.disable: on


[root@gfs01 ~]# mount -t glusterfs 192.168.0.112:test-disperse /mnt/test-disperse
[root@gfs01 ~]# touch /mnt/test-disperse/1.txt
[root@gfs01 ~]# date >> /mnt/test-disperse/1.txt
[root@gfs01 ~]# date >> /mnt/test-disperse/1.txt
...
[root@gfs01 ~]# cat /mnt/test-disperse/1.txt
Wed May 26 11:42:49 EDT 2021
Wed May 26 11:42:49 EDT 2021
Wed May 26 11:42:50 EDT 2021
Wed May 26 11:42:51 EDT 2021
Wed May 26 11:42:51 EDT 2021

[root@gfs01 ~]# md5sum  /mnt/test-disperse/1.txt
95af6ad88c70c127faee8d84e3eb8d8f  /mnt/test-disperse/1.txt
[root@gfs01 ~]# md5sum  /glusterfs/test-disperse/1.txt
0215b19719d86fd12a973926ec0c0854  /glusterfs/test-disperse/1.txt


[root@gfs02 ~]#  md5sum  /glusterfs/test-disperse/1.txt
cdcf65a53fef2a8066bd96d6324dab85  /glusterfs/test-disperse/1.txt

[root@gfs03 ~]#  md5sum  /glusterfs/test-disperse/1.txt
d76f9fc89d264d19641ec552f600d46a  /glusterfs/test-disperse/1.txt
```

我们查看一下每一个 brick 下面的数据情况可以发现文件的 md5 都不一样的，那么这里并不是和完整的挂载目录文件，同时如果感兴趣，这里可以查看一下几个 brick 下面的数据分布情况。

1.2.2.3. force 的作用

前面提到了两种常见的 volume,那么对于 volume 的操作有 status,start 和 info 这些常见的，而其中对于 start 和 stop 都有一个可选项 force，这个的作用其实是什么呢？这里我们可以简单测试一下。

```
[root@gfs01 ~]# gluster volume start

Usage:
```

```
volume start <VOLNAME> [force]

[root@gfs01 ~]# gluster volume stop

Usage:
volume stop <VOLNAME> [force]
```

下面进行简单的测试，方法就是 kill 掉其中一个 brick 的目录，然后重新启动查看一下作用。

```
Status of volume: test-replica
Gluster process                                 TCP Port  RDMA Port  Online  Pid
------------------------------------------------------------------------------
Brick 192.168.0.110:/glusterfs/test-replica 49153     0          Y       1776
Brick 192.168.0.111:/glusterfs/test-replica 49153     0          Y       1620
Brick 192.168.0.112:/glusterfs/test-replica 49155     0          Y       1521
Self-heal Daemon on localhost               N/A       N/A        Y       1160
Self-heal Daemon on gfs02                   N/A       N/A        Y       1426
Self-heal Daemon on gfs03                   N/A       N/A        N       N/A

Task Status of Volume test-replica
------------------------------------------------------------------------------
There are no active volume tasks

//这里删掉了其中一个 brick 目录
[root@gfs01 ~]# rm -fr /glusterfs/test-replica
[root@gfs01 ~]# gluster volume start  test-replica
volume start: test-replica: failed: Failed to find brick directory /glusterfs/test-replica for volume test-replica. Reason : No such file or directory

//前面无法正常启动了.使用 force 强制启动.
[root@gfs01 ~]# gluster volume start  test-replica  force
volume start: test-replica: success
[root@gfs01 ~]# gluster volume status     test-replica
Status of volume: test-replica
Gluster process                                 TCP Port  RDMA Port  Online  Pid
------------------------------------------------------------------------------
Brick 192.168.0.110:/glusterfs/test-replica N/A       N/A        N       N/A
Brick 192.168.0.111:/glusterfs/test-replica 49154     0          Y       1674
Brick 192.168.0.112:/glusterfs/test-replica 49154     0          Y       1573
Self-heal Daemon on localhost               N/A       N/A        Y       1160
Self-heal Daemon on gfs03                   N/A       N/A        N       N/A
Self-heal Daemon on gfs02                   N/A       N/A        Y       1426

Task Status of Volume test-replica
```

```
35. ------------------------------------------------------------------------------
36. There are no active volume tasks
```

从这里可以看到，其中所谓的 force 就是在一些 brick 坏掉异常的时候，能够强制启动，对于 3 副本的分布式复制卷来说，正常是只要超过 2 个 brick 正常就可以启动的，但是默认是不行，需要使用 force 命令。另外这里如果感兴趣，读者可以自行测试一下，前面删掉的数据目录，如果再次重新创建，能否继续生效呢？ 另外这里 force 的作用可以从代码中找到答案。

```
1.  //代码文件路径: xlator->mgmt->glusterd->src->glusterd-volume-ops.c
2.  //函数名: glusterd_op_stage_start_volume
3.
4.  ret = gf_lstat_dir(brickinfo->path, NULL);
5.          if (ret && (flags & GF_CLI_FLAG_OP_FORCE)) {
6.              continue;
7.          } else if (ret) {
8.              len = snprintf(msg, sizeof(msg),
9.                             "Failed to find "
10.                            "brick directory %s for volume %s. "
11.                            "Reason : %s",
12.                            brickinfo->path, volname, strerror(errno));
13.             if (len < 0) {
14.                 strcpy(msg, "<error>");
15.             }
16.             goto out;
17.         }
```

1.2.3. 仲裁节点 arbiter

1.2.3.1. arbiter 到底是什么

在分布式复制卷这里,还有一种特殊的类型，就是仲裁节点,首先向说明一下，为什么需要考虑仲裁节点 arbiter 呢？最大的原因就是为了减少脑裂的发生.(所谓的脑裂现象其实副本中的元数据信息在机器产生故障或者反复宕机后等复杂环境下，元数据信息不一致产生的，在分布式系统中属于非常致命且危险的现象)。为了进一步感受一下仲裁节点的作用，下面先简单创建一个进行测试。

```
1.  [root@gfs03 ~]# gluster volume create test-arbiter replica 2 arbiter 1 192.168.0.{110,111,112}:/glusterfs/test-arbiter f
    orce
2.  volume create: test-arbiter: success: please start the volume to access data
3.  [root@gfs03 ~]# gluster volume start test-arbiter
4.  volume start: test-arbiter: success
5.  [root@gfs03 ~]# gluster volume info test-arbiter
6.
```

```
Volume Name: test-arbiter
Type: Replicate
Volume ID: c0f1d50a-0060-456f-a82e-64fb8b99c020
Status: Started
Snapshot Count: 0
Number of Bricks: 1 x (2 + 1) = 3
Transport-type: tcp
Bricks:
Brick1: 192.168.0.110:/glusterfs/test-arbiter
Brick2: 192.168.0.111:/glusterfs/test-arbiter
Brick3: 192.168.0.112:/glusterfs/test-arbiter (arbiter)
Options Reconfigured:
cluster.granular-entry-heal: on
storage.fips-mode-rchecksum: on
transport.address-family: inet
nfs.disable: on
performance.client-io-threads: off
```

这里注意一点，为了保留一下原来的写法，可能会见到创建文件时写 replica 3 arbiter 1 也是一样的，对于 3 副本的仲裁节点来说，默认都是最后一个 brick 作为仲裁节点 arbiter,而如果是 2*3 的 volume，则是每一组的最后一个，也是第三个 brick 作为 arbiter。同时,作为仲裁节点，那么该 brick 的最大的作用就是对数据元信息的保存，并且不保存实际的数据。

```
[root@gfs03 ~]# mkdir -p /mnt/test-arbiter
[root@gfs03 ~]# mount -t glusterfs 192.168.0.110:test-arbiter /mnt/test-arbiter
[root@gfs03 ~]# dd if=/dev/zero of=/mnt/test-arbiter/1.txt bs=64k count=1000
1000+0 records in
1000+0 records out
65536000 bytes (66 MB) copied, 0.566256 s, 116 MB/s

[root@gfs03 ~]# du -sh /glusterfs/test-arbiter/
16K     /glusterfs/test-arbiter/

[root@gfs02 ~]# du -sh /glusterfs/test-arbiter/
63M     /glusterfs/test-arbiter/

[root@gfs01 ~]# du -sh /glusterfs/test-arbiter/
63M     /glusterfs/test-arbiter/
```

对于仲裁节点来说,因为不直接存放数据,因此这里的磁盘使用量会比其他数据节点少很多,那么关于这个 brick 的目录容量大小估计,这里最好还是根据官方的建议,是复制副本中文件数的 4KB 倍.当然这里的估计值还取决于底层文件系统为给定磁盘大小分配的 inode 空间。

对于大小为 1TB 到 50TB 的卷，XFS 中的 maxpct 值仅为 5%。如果你想存储 3 亿个文件，4kbx300m 给我们 1.2TB。其中 5%的容量在 60GB 左右。假设建议的 inode 大小为 512 字节，那么我们只能存储 60GB/512~=1.2 亿个文件。因此，在格式化大于 1TB 的 XFS 磁盘时，最好选择更高的 maxpct 值（例如 25%）。这里 maxpct 是 xfs 中关于设置 inode 的一个参数，感兴趣的可以自行查阅。

那么关于 arbiter 的性能问题,还做了一个简单的测试,关于 arbiter 和非 arbiter 的 volume 性能.当然这里的测试只是为了测试出相对大小的差异.如果想真正测试出极限情况，建议在测试集群中进一步测试压测。下面先列出两个用于测试的 volume。

```
Volume Name: test-replica
Type: Replicate
Volume ID: 4568c063-5b75-4304-98f6-21f3955cc138
Status: Started
Snapshot Count: 0
Number of Bricks: 1 x 3 = 3
Transport-type: tcp
Bricks:
Brick1: 192.168.0.110:/glusterfs/test-replica
Brick2: 192.168.0.111:/glusterfs/test-replica
Brick3: 192.168.0.112:/glusterfs/test-replica
Options Reconfigured:
performance.client-io-threads: off
nfs.disable: on
transport.address-family: inet
storage.fips-mode-rchecksum: on
cluster.granular-entry-heal: on


Volume Name: test-arbiter
Type: Replicate
Volume ID: c0f1d50a-0060-456f-a82e-64fb8b99c020
Status: Started
Snapshot Count: 0
Number of Bricks: 1 x (2 + 1) = 3
Transport-type: tcp
Bricks:
Brick1: 192.168.0.110:/glusterfs/test-arbiter
Brick2: 192.168.0.111:/glusterfs/test-arbiter
Brick3: 192.168.0.112:/glusterfs/test-arbiter (arbiter)
Options Reconfigured:
performance.client-io-threads: off
nfs.disable: on
transport.address-family: inet
storage.fips-mode-rchecksum: on
cluster.granular-entry-heal: on
```

下面是测试的脚本,这里的测试是在一台客户端的虚拟机中挂载两个 volume,然后执行脚本进行测试的.测试的命令则是使用了常见的 dd 命令.每次创建一个 655MB 的文件,然后查看创建的速度。

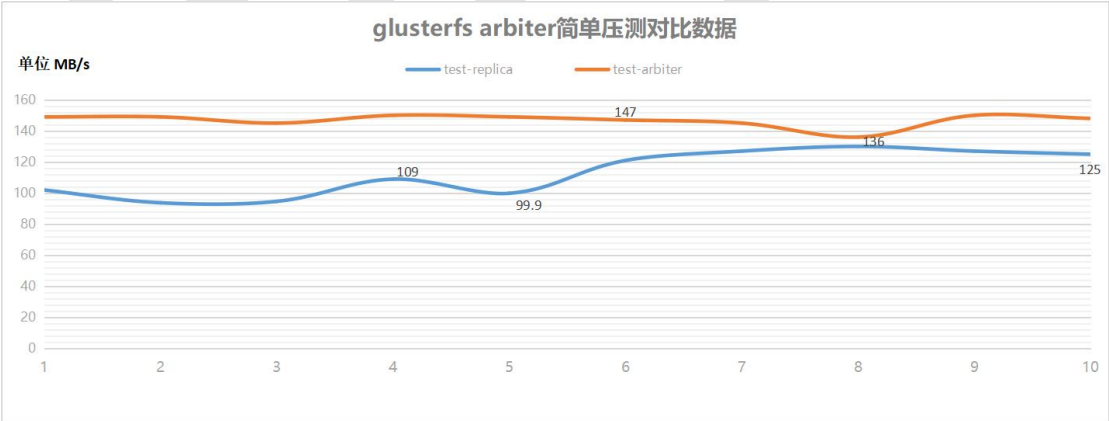
```
#!/bin/bash

#path=$1


test(){
  sudo echo -e "\n path is $1"
 for((i=1;i<=10;i++));
 do
   sudo dd if=/dev/zero of=$1/$i.txt  bs=64k count=10000
   sudo echo -e "\n"
   sudo sleep 2
 done
}

test  /mnt/test-replica
test  /mnt/test-arbiter
```

最后给出测试结果，这里的测试结果表明,arbiter 的 volume 在较大文件的写入时，速度会比常用的 3 副本还要快一些，当然这是因为 arbiter 的 brick 并不承担保存源数据。因此这里的测试可以作为一个业务场景的参考。



这里的测试结果是在 centos7.9 虚拟机上面测试的,而在真实的生产环境中，除了测试这种较大的文件以外，还应该测试一些小文件的读写情况，可以使用以下脚本进行测试。该 脚本就是随机成为一万个大小随机的小文件进行写入测试。

```
#!/bin/bash

test(){
   sudo echo -e "\n\n\n path is: $1"
```

```
    #随机数
    sudo date
    for i in $(seq 1 10000)
    do
      num=`sudo echo $[$RANDOM%100+1]`
      sudo dd  if=/dev/zero of=$1/$i.txt bs=6k count=$num > /dev/null 2>&1
    done
    sudo date
}

test /mnt/test-arbiter/
test /mnt/test-replica/
```

最后关于 arbiter,会有客户端和服务端两个区别,同时每一种还有对应的参数进行选择,感兴趣的建议在测试集群中进行相应的测试。

1.2.3.2. arbiter 的客户端与服务端区别

关于仲裁节点，官方提供了两种不同的模式，一种是客户端，另外一种是服务端，对于两者的区别，下面先简单介绍一下。

仲裁节点类型	服务端	客户端
适用 volume	所有卷	复制卷和分布式复制卷
重要参数	server-quorum-type cluster.server-quorum-ratio	cluster.quorum-type cluster.quorum-count
特点	1. 在 2 副本的 2 个节点的集群上，设置 ratio 超过 51%是不会生效的，没法防止脑裂。 2. 如果不满足多数 bricks 活跃，那么服务端会 kill 掉 bricks 进程，让 volume 无法读取。对防止脑裂的效果会比客户端模式更有效。	1. 当 quorum-type 为 fixed 时，只有当活跃的 bricks 数量超过 quorum-count 时才允许写入。 2. 当 quorum-type 为 auto 时，只有超过半数活跃的 bricks 才能写入。 3. 当活跃的 bricks 恰好为一半时，第一个 bricks 必须活跃。

1.2.4. 集群的相关日志文件

这里主要想简单讲解一下一些常见的默认的集群日志文件目录存放地方，这样对于以后排查日志的时候会比较方便快速地找到对应的内容进行查看。glusterfs 集群的默认日志路径都是在/var/log/glusterfs 下面的,其中这里分为几大类的日志。

日志路径	作用
/var/log/glusterfs/glusterd.log	glusterd 进程日志,该进程是管理每个节点上服务的主进程.
/var/log/glusterfs/cli.log	glusterfs client 在 server 端的日志
/var/log/glusterfs/cmd_history.log	集群执行相关命令的结果历史记录日志,包括执行 status 等.
/var/log/glusterfs/bricks/<path extraction of brick path>.log	每个 brick 在该节点的相关日志
/var/log/glusterfs/VOLNAME-rebalance.log	volume 进程 rebalance 重平衡操作的日志,当 volume 进行扩容或者缩容时会触发 rebalance.
/var/log/glusterfs/glustershd.log /var/log/glusterfs/glfsheal-VOLNAME.log	集群 volume 自愈时产生的信息日志文件,当节点宕机后重新加入集群，数据需要校验恢复,就会触发 heal 功能.
/var/log/glusterfs/quotad.log /var/log/glusterfs/quota-crawl.log /var/log/glusterfs/quota-mount- VOLNAME.log	quota 功能相关日志，就是对 volume 的数据容量大小进行限制的
/var/log/glusterfs/nfs.log	glusterfs nfs 功能日志
/var/log/glusterfs/geo-replication	和不同 glusterfs 集群之间进行数据迁移有关的功能日志

这里实际上还有其他不同的日志，当然有些可能会比较少用到，如果有需要的话，可以到官方文档中进行查看 glusterfs 不同日志文件[2]。另外这里除了和集群有关的日志文件以外，如果是使用了 k8s 集群的话，还要关注以下几个目录下的文件，注意这里是在 k8s 集群节点上的日志文件，并不是在 glusterfs 集群上的。

日志路径	作用
/var/lib/kubelet/plugins/kubernetes.io/glusterfs	这里是 k8s 下的不同存储下的 pvc 日志,有时客户端报错的日志可以在这里查看
/var/log/syslog	k8s 的日志文件信息默认输出到系统日志中

最后,如果想调整一下 volume 的日志文件级别, 可以修改以下两个参数, 设置为 DEBUG 级别的话，会有更加详细的日志输出信息。