

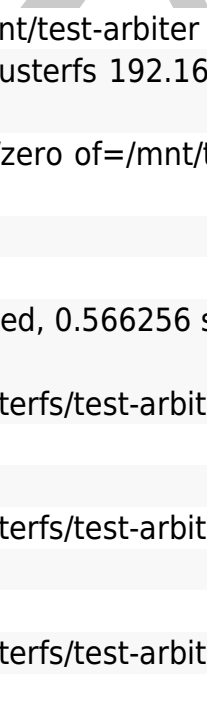
1.2.3. 仲裁节点 arbiter

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 force
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.
7. Volume Name: test-arbiter
8. Type: Replicate
9. Volume ID: c0f1d50a-0060-456f-a82e-64fb8b99c020
10. Status: Started
11. Snapshot Count: 0
12. Number of Bricks: 1 x (2 + 1) = 3
13. Transport-type: tcp
14. Bricks:
15. Brick1: 192.168.0.110:/glusterfs/test-arbiter
16. Brick2: 192.168.0.111:/glusterfs/test-arbiter
17. Brick3: 192.168.0.112:/glusterfs/test-arbiter (arbiter)
18. Options Reconfigured:
19. cluster.granular-entry-heal: on
20. storage.fips-mode-rchecksum: on
21. transport.address-family: inet
22. nfs.disable: on
23. 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。



```
1. [root@gfs03 ~]# mkdir -p /mnt/test-arbiter
2. [root@gfs03 ~]# mount -t glusterfs 192.168.0.110:test-arbiter /mnt/test-arbiter
3. [root@gfs03 ~]# dd if=/dev/zero of=/mnt/test-arbiter/1.txt bs=64k count=1000
4. 1000+0 records in
5. 1000+0 records out
6. 65536000 bytes (66 MB) copied, 0.566256 s, 116 MB/s
7.
8. [root@gfs03 ~]# du -sh /glusterfs/test-arbiter/
9. 16K    /glusterfs/test-arbiter/
10.
11. [root@gfs02 ~]# du -sh /glusterfs/test-arbiter/
12. 63M    /glusterfs/test-arbiter/
13.
14. [root@gfs01 ~]# du -sh /glusterfs/test-arbiter/
15. 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，4kx300m 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 $60\text{GB}/512 \approx 1.2$ 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。

1. Volume Name: test-replica
2. Type: Replicate
3. Volume ID: 4568c063-5b75-4304-98f6-21f3955cc138
4. Status: Started
5. Snapshot Count: 0
6. Number of Bricks: $1 \times 3 = 3$
7. Transport-type: tcp
8. Bricks:
9. Brick1: 192.168.0.110:/glusterfs/test-replica
10. Brick2: 192.168.0.111:/glusterfs/test-replica
11. Brick3: 192.168.0.112:/glusterfs/test-replica
12. Options Reconfigured:
13. performance.client-io-threads: off
14. nfs.disable: on
15. transport.address-family: inet
16. storage.fips-mode-rchecksum: on
17. cluster.granular-entry-heal: on
- 18.
- 19.
20. Volume Name: test-arbiter

21. Type: Replicate
22. Volume ID: c0f1d50a-0060-456f-a82e-64fb8b99c020
23. Status: Started
24. Snapshot Count: 0
25. Number of Bricks: $1 \times (2 + 1) = 3$
26. Transport-type: tcp
27. Bricks:
28. Brick1: 192.168.0.110:/glusterfs/test-arbiter
29. Brick2: 192.168.0.111:/glusterfs/test-arbiter
30. Brick3: 192.168.0.112:/glusterfs/test-arbiter (arbiter)
31. Options Reconfigured:
32. performance.client-io-threads: off
33. nfs.disable: on
34. transport.address-family: inet
35. storage.fips-mode-rchecksum: on
36. cluster.granular-entry-heal: on

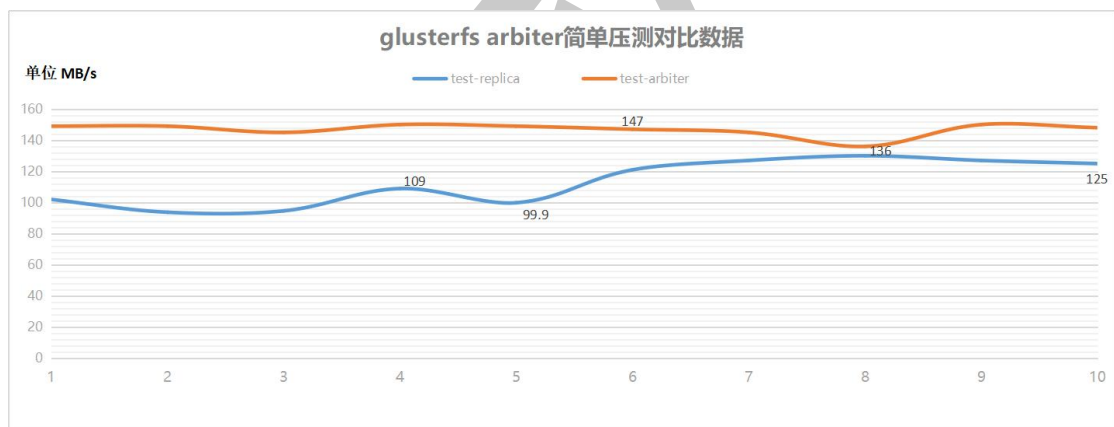
下面是测试的脚本,这里的测试是在一台客户端的虚拟机中挂载两个 volume,然后执行脚本进行测试的.测试的命令则是使用了常见的 dd 命令.每次创建一个 655MB 的文件,然后查看创建的速度。

```

1.  #!/bin/bash
2.
3.  #path=$1
4.
5.
6.  test(){
7.      sudo echo -e "\n path is $1"
8.      for((i=1;i<=10;i++));
9.      do
10.         sudo dd if=/dev/zero of=$1/$i.txt bs=64k count=10000
11.         sudo echo -e "\n"
12.         sudo sleep 2
13.     done
14. }
15.
16. test /mnt/test-replica
17. 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 脚本就是随机生成一万个大随机的小文件进行写入测试。

```
1. #!/bin/bash
2.
3. test(){
4.     sudo echo -e "\n\n path is: $1"
5.     #随机数
6.     sudo date
```

```

7.   for i in $(seq 1 10000)
8.   do
9.       num=`sudo echo [$RANDOM%100+1]`
10.      sudo dd if=/dev/zero of=$1/$i.txt bs=6k count=$num > /dev/nu
      || 2>&1
11.      done
12.      sudo date
13.  }
14.
15.  test /mnt/test-arbiter/
16.  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,另外一种和服务端,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。