GlusterFS: Changing bricks without downtime or data loss

140139

21. november 2014

Sammendrag

This project presents a method to replace a live brick in a running Gluster volume without having downtime nor losing any data. It also explores a way to avoid system overload, by pushing the healing operation to be done when the system load is lower.

Innhold

1 Introduction

GlusterFS is an open source distributed file system with huge a scalability infrastructure, allowing up to 72 brontobytes of data [3]. Gluster volumes are a collection of bricks, it is where most of the configurations happen. A brick is an export directory inside a server, it is the basic storage unit [2]. Inside a volume, the brick is defined by 'server:/path_to_brick'.

With this project I aim to show how to replace and expand a GlusterFS volume without having downtime and avoiding slowness of the system. The scenario where the experiments are running consists of 4 virtual machines running Ubuntu 14.04, 2 Gigabytes of RAM and 2 single core CPUs at 2,1 GHz. Each of these machines got two extra block devices that are utilized as bricks, each one having 50 Gigabytes.

2 SLA/RECSPEC

By doing this project I wish to learn as much as possible about GlusterFS and file systems in general. I see this as an opportunity to acquire practical knowledge, in a real scenario that otherwise would be very hard to get. Also, besides helping me to develop myself as an capable system administrator, this project will have direct positive effect on the university infrastructure, by expanding GlusterFS effective storage space and proving its scalabillity.

To accomplish this, I will start by simulating the real scenario on virtual machines and learning how GlusterFS works. I intend to repeat this process as many times as needed to get a very good understanding of the brick substitution process.

3 Description of your experiment

Before doing any Gluster related operation, it was necessary to create a compatible file system in every brick. The recommended file system by the Gluster team is XFS as it is specially good at parallel operations, due to its design [5]. Also, the recommended inode size is 512. This step was done by running the following commands in each of the nodes.

```
#mkfs.xfs -i size=512 /dev/vdb
#mount /dev/vdb /mnt/brick0/
#mkdir /mnt/brick0/brick
#mkfs.xfs -i size=512 /dev/vdc
#mount /dev/vdc /mnt/brick1/
#mkdir /mnt/brick1/brick
```

The next step consisted on creating the Gluster volume, called CinderSAN, and starting it, from any Gluster server of choice. For this project, all the possible gluster operations were done from *node01*. A directory called *brick* was created inside every mount point, to be used as the path to the bricks in the volume.

```
#gluster volume create CinderSAN replica 2 \
node01:/mnt/brick0/brick node02:/mnt/brick0/brick \
node03:/mnt/brick0/brick node04:/mnt/brick0/brick
```

```
#gluster volume start CinderSAN
```

The option *replica 2* means it will replicate any written data to two disks. After having the volume created and started without errors, the next step is to mount it from the client.

```
#mount -t glustefs node01:/CinderSAN /mnt/gluster/gluster0/
```

For the upcoming operations regarding the brick replacement operations, the CinderSAN volume was filled with 70 GB of data divided in several files. Each of the files had a MD5 checksum associated to it to ensure the data after the replacement is all the same.

Even though Gluster 3.5 has a *replace-brick* option, it is deprecated and the procedure to replace a working brick ends up being the same as replacing a broken one. So, the first step of the procedure is to kill the brick that is going to be replaced. To do so, it is necessary to know which is the brick PID. This information can be obtained by running the following command from any server.

```
#gluster volume status CinderSAN
Status of volume: CinderSAN
Gluster process                            Port    Online   Pid
-----------------------------------------------------------------
Brick 192.168.111.11:/mnt/brick0/brick     49162     Y      8104
Brick 192.168.111.12:/mnt/brick0/brick     49163     Y      4068
Brick 192.168.111.13:/mnt/brick0/brick     49153     Y      4211
Brick 192.168.111.14:/mnt/brick0/brick     49153     Y      4228
...
```

The brick to be killed here is the one from *node01*, represented by the IP address ending in 11. To make sure the volume is not going to have any downtime, a few processes that constantly read and write data in the volume, from the client side, were triggered before killing the brick.

```
#kill -9 8104
#gluster volume status CinderSAN
Status of volume: CinderSAN
Gluster process                            Port    Online   Pid
-----------------------------------------------------------------
Brick 192.168.111.11:/mnt/brick0/brick      N/A      N      8104
```

From now on, the new brick should be mounted and have the directory that is going to be used as the brick path created. For this project, this step was done in the beginning, at the same time as the current bricks were created.

Now it is necessary to set up metadata so the heal from the replica pair will happen. This is done by creating a new directory in the new brick mount point, */mnt/brick1/* in this case, and deleting it, followed by setting *setfattr* and removing it, just like the directory.

```
#mkdir /mnt/brick1/newDirectory
#rmdir /mnt/brick1/newDirectory
#setfattr -n trusted.non-existent-key -v abc /mnt/brick1
#setfattr -x trusted.non-existent-key /mnt/brick1
```

The *setfattr* is necessary to mark the pending changelog wich will trigger the self-heal daemon to heal the new brick with the old brick data. The last step to finish replacing the brick is to use the *replace-brick* operation as follows.

```
#gluster volume replace-brick CinderSAN \
node01:/mnt/brick0/brick node01:/mnt/brick1/brick \
commit force
volume replace-brick: success: replace-brick commit successful
```

Running *status* operation shows the new brick taking place.

```
#gluster volume status CinderSAN
Status of volume: CinderSAN
Gluster process                           Port    Online   Pid
---------------------------------------------------------------
Brick 192.168.111.11:/mnt/brick1/brick    49163     Y      9332
```

Another viable option is to skip the step where the metadata is set to purposely not trigger the self-heal daemon, so it can be triggered manually when the servers are under minimum load by running *gluster volume heal* VOLUME-NAME *full*. However, by doing this, a window of opportunity to Murphy's Law [1] is opened, in case the running replica stops working.

4 Security aspects

The main point here is to always have a backup beyond the replica when replacing a brick. The reason is very straight forward, as the replica that is still running is not fail-proof and may fail during the heal process.

5 Conclusions

The process to replace a live - or dead - brick turns out to be very simple and seemingly safe. Also there is no major requirements to perform this kind of maintenance. The possibility to do the load intense operations when the system load is low is a great trick up the sleeves of system administrators, minimizing the impact of such operations on clients.

It is worth mentioning that Puppet has a module in Puppet Forge for Gluster [4]. It allows automatic installation and configuration of the servers in a pool, as well as the clients. It also provides means to create, modify and mount Gluster volumes.

Referanser

[1] AVIDOR, R. Murphy laws site - murphy laws, 2014. [Online; accessed 19-November-2014]. 3

[2] GLUSTERFS. Glossary - gluster, 2014. [Online; accessed 11-November-2014]. 1

[3] GLUSTERFS. Gluster, 2014. [Online; accessed 11-November-2014]. 1

[4] MEDS, C. M. Gluster puppet module, 2014. [Online; accessed 21-November-2014]. 3

[5] SGI. Xfs: A high-performance journaling filesystem, 2012. [Online; accessed 16-November-2014]. 1