RED HAT STORAGE REPLICATION MECHANISM AND SPLIT-BRAIN SCENARIOS

PATRIC UEBELE / VIJAY BELLUR

Document History

Version	Revision Date	Contributor	Revision Description
1.0	12/09/12	Patric Uebele / Vijay Bellur	Initial version

TABLE OF CONTENTS

REPLICATED VOLUMES IN RED HAT STORAGE 2.0

Replicated volumes replicates files across bricks in the volume. You can use replicated volumes in environments where high-availability and high-reliability are critical. The number of bricks should be equal to the replica count for a replicated volume (or a multiple of the replica count, resulting in a distributed-replicated volume). To protect against server and disk failures, it is recommended that the bricks of the volume are from different servers.

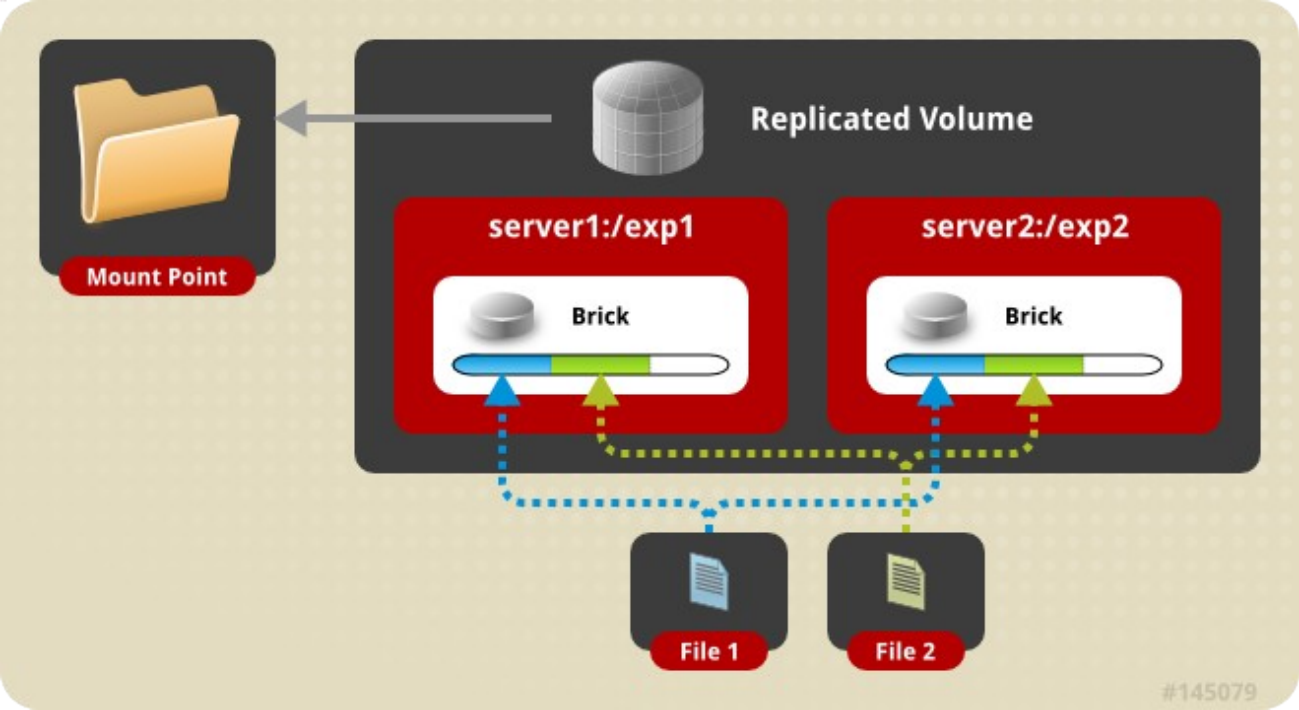

Illustration of a replicated volume (replica count of 2)

Replication Algorithm in Red Hat Storage 2.0

Replication is the most necessarily complex part of Red Hat Storage (RHS), and one of its key differentiators from other solutions. In RHS, the approach is based on using a changelog to mark files as potentially “dirty” while they’re being modified, so that they can be recovered if the modification fails on one replica in the middle of a write or if one of the bricks is not available during a write operation to a file in a replicated volume.

Each file and directory has a changelog which is implemented by a collection of extended attributes. For every replica, the extended attribute **trusted.afr.<VOLUME>-<CLIENT>** is being used to encode the number of pending operations on a file or directory. E.g.

```
[root@server1 export1]# getfattr -m. -d -e hex /export1/splitfile
getfattr: Removing leading '/' from absolute path names
# file: export1/splitfile
trusted.afr.repvol1-client-0=0x000000000000000000000000
trusted.afr.repvol1-client-1=0x000000000000000000000000
trusted.gfid=0x03d37449ee9e4950932b40d87e445100
```

```
[root@server2 export1]# getfattr -m. -d -e hex /export1/splitfile
getfattr: Removing leading '/' from absolute path names
# file: export1/splitfile
trusted.afr.repvol1-client-0=0x000000000000000000000000
trusted.afr.repvol1-client-1=0x000000000000000000000000
trusted.gfid=0x03d37449ee9e4950932b40d87e445100
```

The trusted.afr atribute represents three 4 bytes counters for pending data, metadata and directory entry operations. Each replica contains a changelog for every other replica, so that a failure of one replica does not wipe out both the write and its log. For pending data writes, while **brick-0** was **not** available, the resulting **pending matrix** looks like:

Brick-0	Data	Metadata	Entry
Brick-0	0x00000000	0x00000000	0x00000000
Brick-1	0x00000000	0x00000000	0x00000000

Brick-1	Data	Metadata	Entry
Brick-0	0x00000012	0x00000000	0x00000000
Brick-1	0x00000000	0x00000000	0x00000000

The Conceptual Process of a Write

The process of writing to a file on a replicated volume consists of the following steps:

1. Take a lock on the byte-range of the file to be written to on all the replicas. This prevents simultaneous updates from happening at the replicas in different orders, which is hard to reconcile.
2. Increment the changelog (extended attribute trusted.afr) of the file to reflect ongoing write. Please note that each replica contains a changelog for every other replica.
3. Perform the write(s) on all the replicas that are up.
4. After completion of each replica write, decrement the changelog on the other replica(s).
5. After all writes and changelog decrements are complete on all available replicas, release the lock on all the replicas and acknowledge the write to the upper layer/client.

If there are any unfinished writes, e.g. because one of the replicas wasn't available during a write operation, this is kept recorded in the trusted.afr attributes as in the example below (server2 was down during the write on file "splitfile"):

```
[root@server1 export1]# getfattr -m. -d -e hex /export1/splitfile
getfattr: Removing leading '/' from absolute path names
# file: export1/splitfile
```

```
trusted.afr.repvol1-client-0=0x000000000000000000000000
trusted.afr.repvol1-client-1=0x000000070000000000000000
trusted.gfid=0x03d37449ee9e4950932b40d87e445100
```

```
[root@server2 export1]# getfattr -m. -d -e hex /export1/splitfile
getfattr: Removing leading '/' from absolute path names
# file: export1/splitfile
trusted.afr.repvol1-client-0=0x000000070000000000000000
trusted.afr.repvol1-client-1=0x000000000000000000000000
trusted.gfid=0x03d37449ee9e4950932b40d87e445100
```

Additionally, each brick maintains an index of files and directories that have trusted.afr bits set in /<BRICK>/.glusterfs/indices/xattrop, e.g.

```
[root@server1 xattrop]# ls -l /export1/.glusterfs/indices/xattrop
total 0
---------- 23 root root 0 Aug  7 11:47 00000000-0000-0000-0000-000000000001
---------- 23 root root 0 Aug  7 11:47 00a8c854-80bd-4793-87cb-165e13103d58
---------- 23 root root 0 Aug  7 11:47 0d86381d-a111-4c63-a973-f054255f8508
---------- 23 root root 0 Aug  7 11:47 1bee865f-ce6e-4ab9-bf17-69a71f7f9291
---------- 23 root root 0 Aug  7 11:47 4cfbcbd3-166e-4707-8cef-14cbd8112daa
---------- 23 root root 0 Aug  7 11:47 6872f1c9-b573-4e5e-917c-0c26f7ea3298
....
```

SELF-HEALING

Once a replica with outstanding writes to it is up again, files with pending writes have to be synchronized in order to maintain data redundancy and consistency. This is done automatically by the so-called **self-heal daemon**, which runs in the background and checks every 10 minutes for issues and triggers self-healing on the entire volume or only on the files that need healing. A self-heal can also be triggered manually with the “gluster volume heal” command. This triggers an index based self-heal, querying the index to perform self-heal on the files listed in the index. A full self-heal (gluster volume heal <VOLUME> full) traverses the whole brick to perform self-heal, checking the trusted.afr attributes of all files on the brick.

The direction of synchronization is based on the values of the changelogs in the trused.afr attributes on the replicas. Based on the value of the changelog entries, we give the replicas the following “characters”:

- IGNORANT means that the replica doesn’t have a changelog for a file, e.g. for a file that is missing on one replica.
- INNOCENT means that the replica does have only zero entries in its changelog.
- FOOL means that the replica has non-zero entries in the changelog for itself. In other words, it got as far as the changelog increment but not as far as the decrement, so we don’t actually know whether the write in between made it to disk.
- WISE means that the replica has non-zero entries in the changelog for the other replica, while having only zeroes

for itself.

Brick-0 (INNOCENT)	Data	Metadata	Entry
Brick-0	0x00000000	0x00000000	0x00000000
Brick-1	0x00000000	0x00000000	0x00000000

Brick-0 (FOOL)	Data	Metadata	Entry
Brick-0	0x00000001	0x00000000	0x00000000
Brick-1	Don’t care	Don’t care	Don’t care

Brick-0 (WISE)	Data	Metadata	Entry
Brick-0	0x00000000	0x00000000	0x00000000
Brick-1	0x00000011	0x00000000	0x00000000

Based on the characters of the replicas of a volume, the source for the self-heal process is chosen. Without going into too much detail of the algorithm, the most common scenarios are:

- All replicas are INNOCENT. Data and meta-data self-heals get triggered. RHS tries to be conservative here. The file with smallest UID is picked up as source for meta-data healing and file with size 0 as the destination for data self heal.
- Only one WISE replica exists. The single WISE node will be the source for the self-heal and its data will be propagated to the other replicas
- Multiple WISE replicas exist, and they have non-zero entries in their changelogs for the other replicas. This is the infamous “split brain” situation where modifications to a file were made on both replicas while there was no communication between the replicas. This situation can not be resolved automatically. We will deal with it in a bit more detail in the next section.

SPLIT-BRAIN SCENARIO

In a (data) split-brain situation, the characters of the replicas look like this:

Brick-0 (WISE)	Data	Metadata	Entry
Brick-0	0x00000000	0x00000000	0x00000000
Brick-1	0x0000011	0x00000000	0x00000000

Brick-1 (WISE)	Data	Metadata	Entry
Brick-0	0x0000011	0x00000000	0x00000000
Brick-1	0x00000000	0x00000000	0x00000000

and the extended attribute values on the replica bricks will look as in the example below:

```
[root@server1 export1]# getfattr -m. -d -e hex /export1/splitfile
getfattr: Removing leading '/' from absolute path names
# file: export1/splitfile
trusted.afr.repvol1-client-0=0x000000000000000000000000
trusted.afr.repvol1-client-1=0x000000110000000000000000
trusted.gfid=0x03d37449ee9e4950932b40d87e445100

[root@server2 export1]# getfattr -m. -d -e hex /export1/splitfile
getfattr: Removing leading '/' from absolute path names
# file: export1/splitfile
```

```
trusted.afr.repvol1-client-0=0x000000110000000000000000
trusted.afr.repvol1-client-1=0x000000000000000000000000
trusted.gfid=0x03d37449ee9e4950932b40d87e445100
```

The self-heal daemon will detect this split-brain situation and report errors in the log file of the self-heal daemon /var/log/glusterfs/glustershd.log:

```
[2012-07-30 13:11:16.427137] E [afr-self-heal-common.c:2156:afr_self_heal_completion_cbk]
0-repvol1-replicate-0: background  data self-heal failed on <gfid:03d37449-ee9e-4950-
932b-40d87e445100>
[2012-07-30 13:11:16.427180] W [afr-self-heal-
data.c:831:afr_lookup_select_read_child_by_txn_type] 0-repvol1-replicate-0:
<gfid:03d37449-ee9e-4950-932b-40d87e445100>: Possible split-brain
```

The gfid value gives you the file that is in a split-brain situation, the files in split-brain can also be retrieved with the gluster volume heal command:

```
[root@server2 export1]# gluster volume heal repvol1 info split-brain
Heal operation on volume repvol1 has been successful

Brick server1:/export1
Number of entries: 0

Brick server2:/export1
Number of entries: 1
at                    path on brick
-----------------------------------
2012-07-30 09:47:37 /splitfile
```

Accessing files in a split-brain state from RHS clients will result in I/O errors as long as the files are in that state.

It's up to the administrator and/or application manager to decide which copy of the files in split-brain is the valid (or more valuable) one. Then remove the "invalid' replicas of the files (directly on its brick filesystem) as well as its corresponding entries in /<BRICK>/.glusters and manually trigger a **full** self-heal run[1]:

```
[root@server1 export1]# gluster volume heal repvol1 full
Heal operation on volume repvol1 has been successful
```

which synchronizes the affected file(s) (see /var/log/glusterfs/glustershd.log):

```
[2012-07-30 14:14:33.840540] I [afr-common.c:1340:afr_launch_self_heal] 0-repvol1-
replicate-0: background  meta-data data self-heal triggered. path: <gfid:03d37449-ee9e-
4950-932b-40d87e445100>, reason: lookup detected pending operations
[2012-07-30 14:14:33.851045] I [afr-self-heal-algorithm.c:116:sh_loop_driver_done] 0-
```

1 Index self-heal is dependent on operations that happen from mount points of a gluster volume. Since the deletion of "bad" replica happens outside the ambit of mount points (directly on a brick), index self-heal cannot heal such a case. The workaround is to either access that file or the parent directory. Gluster volume heal <volname> full acts as a workaround since it does a crawl of the volume accessing all files and directories.

```
repvol1-replicate-0: full self-heal completed on <gfid:03d37449-ee9e-4950-932b-
40d87e445100>
[2012-07-30 14:14:33.853367] I [afr-self-heal-common.c:2159:afr_self_heal_completion_cbk]
0-repvol1-replicate-0: background  meta-data data self-heal completed on <gfid:03d37449-
ee9e-4950-932b-40d87e445100>
```

and the trusted.afr attributes of both replicas are set to zero again:

```
[root@server1 export1]# getfattr -m. -d -e hex /export1/splitfile
getfattr: Removing leading '/' from absolute path names
# file: export1/splitfile
trusted.afr.repvol1-client-0=0x000000000000000000000000
trusted.afr.repvol1-client-1=0x000000000000000000000000
trusted.gfid=0x03d37449ee9e4950932b40d87e445100
```

```
[root@server2 export1]# getfattr -m. -d -e hex /export1/splitfile
getfattr: Removing leading '/' from absolute path names
# file: export1/splitfile
trusted.afr.repvol1-client-0=0x000000000000000000000000
trusted.afr.repvol1-client-1=0x000000000000000000000000
trusted.gfid=0x03d37449ee9e4950932b40d87e445100
```

Client access to the affected file(s) should now work again[2].

REFERENCES

- Red Hat Storage 2.0 Administration Guide: https://access.redhat.com/knowledge/docs/en-US/Red_Hat_Storage/2.0/html/Administration_Guide/index.html
- Red Hat Storage 2.0 Release Notes: https://access.redhat.com/knowledge/docs/en-US/Red_Hat_Storage/2.0/html/2.0_Release_Notes/index.html
- Jeff Darcy's blog about replication internals: http://hekafs.org/index.php/2012/03/glusterfs-algorithms-replication-present/

2 If one still gets an I/O error, drop cached information on the client: "echo 3 > /proc/sys/vm/drop_caches"