Automatic File Replication (AFR) in GlusterFS

Vikas Gorur <vikas@zresearch.com>

December 18, 2008

Overview

This document describes the design and usage of the AFR translator in GlusterFS. This document is valid for the 1.4.x releases, and not earlier ones.

The AFR translator of GlusterFS aims to keep identical copies of a file on all its subvolumes, as far as possible. It tries to do this by performing all filesystem mutation operations (writing data, creating files, changing ownership, etc.) on all its subvolumes in such a way that if an operation succeeds on atleast one subvolume, all other subvolumes can later be brought up to date.

In the rest of the document the terms "subvolume" and "server" are used interchangeably, trusting that it will cause no confusion to the reader.

Usage

A sample volume declaration for AFR looks like this:

```
volume afr
 type cluster/afr
 # options, see below for description
 subvolumes brick1 brick2
end-volume
```

This defines an AFR volume with two subvolumes, brick1, and brick2. For AFR to work properly, it is essential that its subvolumes support **extended attributes**. This means that you should choose a backend filesystem that supports extended attributes, like XFS, ReiserFS, or Ext3.

The storage volumes used as backend for AFR *must* have a posix-locks volume loaded above them.

```
volume brick1
 type features/posix-locks
 subvolumes brick1-ds
end-volume
```

Design

Read algorithm

All operations that do not modify the file or directory are sent to all the subvolumes and the first successful reply is returned to the application.

The read() system call (reading data from a file) is an exception. For read() calls, AFR tries to do load balancing by sending all reads from a particular file to a particular server.

The read algorithm is also affected by the option read-subvolume; see below for details.

Classes of file operations

AFR divides all filesystem write operations into three classes:

- **data:** Operations that modify the contents of a file (write, truncate).
- **metadata:** Operations that modify attributes of a file or directory (permissions, ownership, etc.).
- **entry:** Operations that create or delete directory entries (mkdir, create, rename, rmdir, unlink, etc.).

Locking and Change Log

To ensure consistency across subvolumes, AFR holds a lock whenever a modification is being made to a file or directory. By default, AFR considers the first subvolume as the sole lock server. However, the number of lock servers can be increased upto the total number of subvolumes.

The change log is a set of extended attributes associated with files and directories that AFR maintains. The change log keeps track of the changes made to files and directories (data, metadata, entry) so that the self-heal algorithm knows which copy of a file or directory is the most recent one.

Write algorithm

The algorithm for all write operations (data, metadata, entry) is:

1. Lock the file (or directory) on all of the lock servers (see options below).
2. Write change log entries on all servers.
3. Perform the operation.
4. Erase change log entries.
5. Unlock the file (or directory) on all of the lock servers.

The above algorithm is a simplified version intended for general users. Please refer to the source code for the full details.

Self-Heal

AFR automatically tries to fix any inconsistencies it detects among different copies of a file. It uses information in the change log to determine which copy is the "correct" version.

Self-heal is triggered when a file or directory is first "accessed", that is, the first time any operation is attempted on it. The self-heal algorithm does the following things:

If the entry being accessed is a directory:

- The contents of the "correct" version is replicated on all subvolumes, by deleting entries and creating entries as necessary.

If the entry being accessed is a file:

- If the file does not exist on some subvolumes, it is created.
- If there is a mismatch in the size of the file, or ownership, or permission, it is fixed.
- If the change log indicates that some copies need updating, they are updated.

Split-brain

It may happen that one AFR client can access only some of the servers in a cluster and another AFR client can access the remaining servers. Or it may happen that in a cluster of two servers, one server goes down and comes back up, but the other goes down immediately. Both these scenarios result in a "split-brain".

In a split-brain situation, there will be two or more copies of a file, all of which are "correct" in some sense. AFR without manual intervention has no way of knowing what to do, since it cannot consider any single copy as definitive, nor does it know of any meaningful way to merge the copies.

If AFR detects that a split-brain has happened on a file, it disallows opening of that file. You will have to manually resolve the conflict by deleting all but one copy of the file. Alternatively you can set an automatic split-brain resolution policy by using the 'favorite-child' option (see below).

Translator Options

AFR accepts the following options:

read-subvolume (default: none)

The value of this option must be the name of a subvolume. If given, all read operations are sent to only the specified subvolume, instead of being balanced across all subvolumes.

favorite-child (default: none)

The value of this option must be the name of a subvolume. If given, the specified subvolume will be preferentially used in resolving conflicts ("split-brain"). This means if a discrepancy is noticed in the attributes or content of a file, the copy on the 'favorite-child' will be considered the definitive version and its contents will *overwrite* the contents of all other copies. Use this option with caution! It is possible to *lose data* with this option. If you are in doubt, do not specify this option.

Self-heal options

Setting any of these options to "off" prevents that kind of self-heal from being done on a file or directory. For example, if metadata self-heal is turned off, permissions and ownership are no longer fixed automatically.

data-self-heal (default: on)

Enable/disable self-healing of file contents.

metadata-self-heal (default: off)

Enable/disable self-healing of metadata (permissions, ownership, modification times).

entry-self-heal (default: on)

Enable/disable self-healing of directory entries.

Change Log options

If any of these options is turned off, it disables writing of change log entries for that class of file operations. That is, steps 2 and 4 of the write algorithm (see above) are not done. Note that if the change log is not written, the self-heal algorithm cannot determine the "correct" version of a file and hence self-heal will only be able to fix "obviously" wrong things (such as a file existing on only one node).

data-change-log (default: on)

Enable/disable writing of change log for data operations.

metadata-change-log (default: on)

Enable/disable writing of change log for metadata operations.

entry-change-log (default: on)

Enable/disable writing of change log for entry operations.

Locking options

These options let you specify the number of lock servers to use for each class of file operations. The default values are satisfactory in most cases. If you are extra paranoid, you may want to increase the values. However, be very cautious if you set the data- or entry- lock server counts to zero, since this can result in *lost data.* For example, if you set the data-lock-server-count to zero, and

two applications write to the same region of a file, there is a possibility that none of your servers will have all the data. In other words, the copies will be *inconsistent*, and *incomplete*. Do not set data- and entry- lock server counts to zero unless you absolutely know what you are doing and agree to not hold GlusterFS responsible for any lost data.

data-lock-server-count (default: 1)

Number of lock servers to use for data operations.

metadata-lock-server-count (default: 0)

Number of lock servers to use for metadata operations.

entry-lock-server-count (default: 1)

Number of lock servers to use for entry operations.

Known Issues

Self-heal of file with more than one link (hard links):

Consider two servers, A and B. Assume A is down, and the user creates a file 'new' as a hard link to a file 'old'. When A comes back up, AFR will see that the file 'new' does not exist on A, and self-heal will create the file and copy the contents from B. However, now on server A the file 'new' is not a link to the file 'old' but an entirely different file.

We know of no easy way to fix this problem, but we will try to fix it in forthcoming releases.

File re-opening after a server comes back up:

If a server A goes down and comes back up, any files which were opened while A was down and are still open will not have their writes replicated on A. In other words, data replication only happens on those servers which were alive when the file was opened.

This is a rather tricky issue but we hope to fix it very soon.

Frequently Asked Questions

1. How can I force self-heal to happen?

You can force self-heal to happen on your cluster by running a script or a command that accesses every file. A simple way to do it would be:

```
$ ls -lR
```

Run the command in all directories which you want to forcibly self-heal.

2. Which backend filesystem should I use for AFR?

You can use any backend filesystem that supports extended attributes. We know of users successfully using XFR, ReiserFS, and Ext3.

3. What can I do to improve AFR performance?

Try loading performance translators such as io-threads, write-behind, io-cache, and read-ahead depending on your workload. If you are willing to sacrifice correctness in corner cases, you can experiment with the lock-server-count and the change-log options (see above). As warned earlier, be very careful!

4. How can I selectively replicate files?

There is no support for selective replication in AFR itself. You can achieve selective replication by loading the unify translator over AFR, and using the switch scheduler. Configure unify with two subvolumes, one of them being AFR. Using the switch scheduler, schedule all files for which you need replication to the AFR subvolume. Consult unify and switch documentation for more details.

Contact

If you need more assistance on AFR, contact us on the mailing list <gluster-users@gluster.org> (visit gluster.org for details on how to subscribe).

Send you comments and suggestions about this document to <vikas@zresearch.com>.