Distributed Storage: Redundancy

Consensus

**Recovery Mechanism:**

This recovery mechanism we designed is based on log information. The rebooted machine would execute identical commands as other good running machines did and stored in their logs. Different from Raft, which only works as a naming server that doesn’t do the actual storage, our system is designed for real file managing and storage. Therefore, the ability to recover a real file (e.g. image) is essential to our system. The specific operations for every command is as below:

*Upload <Path> <Filename>:* Request file from good running machines and store locally at the correct path

*Rm <Path/Filename>:* Remove local file

*Mkdir <Path/Dir>:* Create a directory locally at the correct path

*Rmdir <Path/Dir>:* Remove local directory

*Mv <srcPath/Filename> <detPath>: Copy local file at srcPath to detPath and remove the local file at srcPath*

*Cp <srcPath/Filename> <detPath>: Copy local file at srcPath to detPath*

Our recovery mechanism would track the log index of good running machine and rebooted machine. The first log to run can be located by comparing the log of rebooted machine and the others. All the commands would be running under the log order one by one. While this simple model could work for most of the situations, we found cases that may cause a trouble:

37: Upload Dir1 image1.jpg

38: Rm Dir1/image1.jpg

If rebooted machine crashed before index 37 and there are two logs in this order in the good running machine, the rebooted machine may meet a trouble when execute command in 37. Since image1.jpg has been deleted on other machines when executed command in 38, request to other machines for image1.jpg can’t retrieve any file back. Ignoring this can be a reasonable solution since the next command in 38 would delete image1.jpg again. However, when we replace ‘38: Rm Dir1/image1.jpg’ with ’38: Mv Dir1/image1.jpg Dir2’, it can’t be ignored or image1.jpg would be lost permanently on the rebooted machine.

Our first approach to solve this problem is to transfer the file to temporary files instead of removing it permanently. Therefore, ‘Rm Dir1/image1.jpg’ would actually change the name of ‘image1.jpg’ into ‘image1.jpg##tmp’. When request from rebooted machine can’t find ‘image1.jpg’, it would retrieve ‘image1.jpg##tmp’ back which has the same content. However, this approach is not perfect. If we have following logs in the good running machine:

37: Upload Dir1 image1.jpg

38: Rm Dir1/image1.jpg

39: Upload Dir1 image1.jpg

40: Mv Dir1/image1.jpg Dir2

and ‘image1.jpg’ are of different content in command 37 and 39, command 38 and command 40 would generate two files with the same name ‘image1.jpg##tmp’, which may confuse the later request for it. Using metadata to separate them could be a solution but a little inefficient. So we use another strategy to solve this problem. Instead of storing the file only by its name, we append “##<index>” to the filename. Therefore, ‘image1.jpg’ in 37 and 39 would be stored as ‘image1.jpg##37’ and ‘image1.jpg##39’ respectively, and their tmp file would be named as ‘image1.jpg##37##tmp’ and ‘image1.jpg##39##tmp’, which solved the mentioned problem.

Furthermore, keeping a large number of obsolete files is space wasting. So we designed a garbage collection mechanism to remove these temporary files when receiving command from the master server at the moment that all the machines are online and up to date.