# Lightweight Distributed File System For Local Networks

SHAH ABUL HASNAT CHOWDHURY, BRAC University, Bangladesh
MD. ESADULHAQ, BRAC University, Bangladesh
TANVIR RAHMAN, BRAC University, Bangladesh
FAHIM FERDOUS, BRAC University, Bangladesh
MD HUMAION KABIR MEHEDI, BRAC University, Bangladesh
MD. MUSTAKIN ALAM, BRAC University, Bangladesh
ANNAJIAT ALIM RASEL, BRAC University, Bangladesh

Starting out with Sun Microsystem's "Network File System", Distributed File Systems (DFS for short) have evolved substantially in the last two decades. But when we look at modern solutions like Hadoop or IPFS, it is immediately apparent that these technologies are solving specific problems. Hadoop, for example is focused on large data reads for analysis and IPFS is there to make the largest file system ever. While there are researches on these large scaled distributed file system, very little thoughts have gone into smaller scaled distributed file systems for simple, lightweight file sharing & plug-and-play uses.

If we were to sit in an office and share some files around, the first option that would come into a programmer's mind is probably *rsync* or *ssh*. But for a regular user (who is not accustomed to these technologies), they might use Samba or Windows Share. While Samba provides interoperability between Linux and Windows, it is still somewhat complicated. Lastly, sharing in Windows is only bounded to the Windows operating system.

When a more robust file sharing mechanism is needed, that is not only simple but also plug-and-play, we could easily do that using a peer-to-peer architecture with the help of a host discovery solution. Also note that we are in an advantage here, since we are in a trusted network. In such a simple file sharing system, every nodes can be discrete since we do not need to provide guarantee of existence for a file, due to the nature of the environment (i.e. there is no file if nobody is sharing it).

For now, we will only talk about read only clients. The client has 6 different components. First let us list the components and then we will describe each of them. Below are the component list sorted by level of abstraction:

- File fingerprint
- File mimetype detection
- Name indexing
- Folder/directory fingerprint
- Consensus mechanism for tracking contents throughout the network
- Graphical user interface (possibly through browser) for interaction

For content addressing, it is useful to identify each file/folder by their content's hash, rather than their name, since names can be different regardless of the contents. It is also useful to have mimetype detection for the users. To make interactions simple, structured and stateless, we will employ a RESTful API for searching and querying. Lastly, we will index the names of the files and directories.

When the program first boots up, it will fetch the machine ID (*/etc/machine-id* in Linux systems) for node identification purposes on the network. Then we start our first task, which is to walk the user specified directory to create a database of files with their name, size, modification/creation time & content hash. While walking, if we are done walking a whole directory, then we append the files hashes, then make a hash out of that hash list. This will be the address of that particular directory. If there are one or more directory, then we simply treat the child directory as another file, and add its hash to the list. Here, we the the list will need to be ordered by *modtime*. In each step of the walk, whenever we are done with a directory, we will add it to our local SQLite database.

Since this is only a read only client, and we are fingerprinting the whole file using 'blake3', if large files were to get modified frequently, then we would run into performance issues. But it is possible

we can add new files to the specified directory, while disallowing modifications. For deletion, it is as simple as deleting the row from the local state database.

As already mentioned before, we can use Multicast DNS for host discovery. After a node has been discovered, the user can connect to that node and browse for files. But when adding new files, we will need a consensus mechanism to determine what files we have on the network itself. This means that, each of the nodes will keep a local databases for each nodes on the network, listing what files a node might have. This being a trusted network, we can use Paxos for the purpose of tracking our contents, i.e. who has what. If we can track this information, then downloading large files would be easier if some other node has that file too. This information will be stored in a file, rather than a SQL database.