CS 244B Project

Rafael Esteves

June 2022

1 Abstract

I have built a distributed file system that allows users to work on a particular file across multiple devices using vim. By tracking the history of a file and sharing that information across the network, I am able to update computers with the most up to date version of a file. While vector time pairs like that used in tra[1] impose no restriction on synchronization patterns, never falsely detect conflicts, require no space to store deletion notices, require network bandwidth proportional only to the number of files changed, and support partial synchronizations; I am able to achieve most of these properties however I introduce other issues.

2 Introduction

Those who use more than one computer must synchronize their files across their network, which keeps the files up to date. Examples of solutions for this type of problem are google drive and github. However, these solutions require that you hand over your files to corporations that are practically unregulated. We design a synchronization system which is hosted entirely on a user's own devices.

Our system uses a history of each file it is tracking to determine the most up to date version. When a user opens a file in vim, the system compares the history of that file for each computer in the network and decides which is the

most recent version, which it displays to the user. Each computer is able to keep track of all of the histories in the network by transferring them using scp.

A previous work, "File Syncronization with Vector Time Pairs" (Cox 2005) lists the properties of an ideal file synchronizer.[1] They impose no restriction on synchronization patterns, meaning that nodes in the network can be updated in any order. They never falsely detect conflicts, require no space to store deletion notices, require network bandwidth proportional only to the number of files changed. Finally, they support partial synchronizations which allows users to synchronize any subset of their files.

While my system is able to achieve many of these properties, it has some other limitations. Firstly, the network bandwidth is proportional to the number of files changed and the length of each file. Another important issue that is unrelated to the above properties is that my system does not ensure a best effort to transport the files as it is built into the saving process and is not retried. If a node in the network is unavailable for some reason when a file is updated, it will have to wait for the next update to receive the changes. All of these issues could have been solved, as is described in the Potential Improvements section.

3 Design

In order to determine the most up to date version of a file a computer in the network must maintain a history of the file on each of the other computers. My system tracks the history of a file by keeping a copy of each iteration of the file. Specifically, before a file is modified I push each version in the history of a file back by one and the new modification becomes the most recent version.

A computer communicates it's history to others in the network by using scp. The user must modify a config file that provides the ssh login information to each of the nodes in the network. From then on whenever the user saves a file a scp command is used to transfer the history of the current file to all other nodes.

When a computer opens a file, the merging algorithm is run. It loops through the computers listed in the config file to check which of them have sent their histories. For each of the histories we compare it with our history and check

for different cases. In case 1, our most recent version of the file and the other computers' most recent version of the file are the same (they contain the same string) in which case we should do nothing for this iteration. In case 2, the other computers' history is a prefix of our history and therefore we should again, do nothing. In case 3, our history is a prefix of the other nodes' history in which case I overwrite our history with their history. A history "A" is a prefix of another history "B" if each version in "A" is also present in "B". Finally, if none of the above are true then there is a conflict which the system handles by creating a new iteration of the file which contains the most recent versions of the file from both computers delimited by a very noticeable line.

4 Potential Improvements

One important improvement I could make to the system is to make it centralized. One limitation of my system is that there is no system to retry transmitting the file. Therefore if a node is off or unreachable for some reason it will never receive an update. A centralized system could retransmit the appropriate files to every node in the network as appropriate. Furthermore, it would allow the non leader nodes to only keep the most up to date version of the file, instead of storing all of the histories for the merging algorithm, reducing the memory requirements. Another relatively simple change that could also reduce computational resource consumption is that histories be stored as the hashes of each iteration of development as opposed to the entire string contained in the file. In fact this would fulfill the property of an ideal file synchronizer that the network bandwidth be proportional only to the number of files modified.

Another important restriction is that my system requires that files be in a very specific format. Each file must be represented by a directory found at $/files/\{file\_name\}/$ and the user must always edit the version1.txt file found in this directory. A solution to this could be to create a new config file which contains a list of the location of all the files in the system and is added to whenever a new file is created.

References

[1] William Josephson Russ Cox. “File Synchronization with Vector Time Pairs”. In: (2005).