CS60 Project: Dartsync

Binjie Li
Jamie Yang
Ben Ribovich
Frederick Ho

06/01/15

1 Introduction

"Dartsync" is our version of Dropbox that circumvents the inconvenience of uploading to a centralized, third-party cloud and instead uses devices to directly talk to each other to obtain the latest version of files in a shared public repository of choice on each device. There are two primary participants in the network structure of Dartsync: the tracker and the peers. The tracker recognizes participating "peers" who want to gain in on this repository shared across the network and keeps track of the most up to date version of the files and on which peers they are located in the shared repository. The peers do the heavy lifting and are responsible for the transmission of the actual files, pinging the tracker at an interval so as to guarantee that the peer does not die.

2 Design Summary

With the basics of Dartsync having already been discussed, we can go into more detail regarding how the work of Dartsync was provisioned.

The four main sections that the project was partitioned into were:

2.1 Peer-to-Peer

2.2 Peer Side

2.3 Tracker Side

The tracker side is composed of a table of alive peers, a table of the most recent version of the files for which Dartsync is running on the peers, and three threads: the **main thread**, the **handshake monitor thread** and the **monitor alive thread**. The basic types of packets recieved from the peer are REGISTER, KEEP ALIVE, and FILE UPDATE.

The tracker itself must be the first node to start running as peers establish themselves as alive according to the tracker before it can partake in the Dartsync filesharing system. On connection, it will send out its own information regarding its IP and the sent time to the tracker. On the tracker side, the tracker's handshake thread will catch this register packet and then send out the current filetable.

At this point, the peer will receive the tracker's filetable and then update its own filetable accordingly. But because peers are obligated to send out FILE UPDATE notifications to the tracker on every modification to the table, we run into the risk of updating the tracker's current file table to what may not be the most recent iteration. For instance, if the tracker file table is updated immediately afterwards with a new file, this file will not exist to the new peer and the FILE UPDATE message that it sends promptly afterwards will delete inadvertently the file in the tracker. To work around this, there is an "isInitialized" flag in the handshake monitor thread that is toggled off at first. Thus when the tracker receives a new peer's first FILE UPDATE packet, it knows to ignore it in terms of the the file delete "changes" that this new peer wants to provide.

With regards to the monitoring of alive peers, the tracker upon receiving the initial REGISTER packet from the peer sends that peer an interval smaller than the minimum amount of time needed for the tracker to acknowledge that peer as alive. The peer then will periodically send KEEP ALIVE messages to guarentee that it stays in the tracker's peer table. If the peer dies or the tracker fails to receive the REGISTER message within the designated time frame for staying alive, then the peer is dropped from the peer table.

2.4 File monitoring & File table structuring

2.4.1 File monitoring

2.4.2 File table structuring

The overall file table data structure is one that is shared on both the peer and tracker side. It consists of a filetable struct that contains information about the number of files, a mutex, a flag for blocking monitor and a linked list for the actual files. The linked list is made out of filetable_entry structs that contains an information struct, the pointer to the next element in a list.

To tackle the issue of keeping the most up to date version of the files, each filetable_entry struct also contains a static array of peers and the total number of peers who have the file. This is an entirely separate list from the tracker's alive peer list and should not be confused with the such. Only peers who definitely contain this file will have their information stored in this list. Upon updates to a filetable_entry, the peerlist will be cleared and only the holding peer's name will be put inside.

Finally, the file information struct discussed earlier contains the filename (restricted to a length), the size of the file in bytes, the timestamp of the last update, and the type.

3 Possible Extensions to the Project

4 Lessions Learned

4.1 Binje

4.2 Jamie

4.3 Ben

4.4 Fred

Start early and start collaborating with others early. Be very careful when handling pointers in threads. There was a small problem with the pointer for the connection number and it was getting changed inadvertently and stumped the process of testing the project for an unnecessarily long period of time.