Michael Jee
Minh Truong
5/1/21
CECS 327

Term Project – File Synchronization

Circular doubly linked list of nodes (Pseudo Chord)

- Each node has 'pointers' which are just ip addresses of its predecessor and successor nodes
- Any time a node changes data in a file, the dht that is generated every X seconds 'updates' so the next node will know there is an update
- Nodes need to constantly poll their predecessor for a connection

Joining the network

- Client node joins a network by using FindIpAddresses and getNodes(). It then chooses a random server node from this list (probably the 0th element since this should be one of the fastest)
    - If there are no other nodes on our port, the client effectively becomes a network of 1
- It connects to this server node
- It sends a message that it wants to join the network
- The server node sends a message to its successor that a new node wants to join the network
- The client node becomes the server node's new successor and the server node becomes the client nodes new predecessor
- The server nodes' old successor becomes the client nodes' new successor and the client node becomes the server node's old successor's predecessor (see appendix figure 1)

Connecting

- Every node always has a server thread running
- Every X seconds, the node's server thread connects to the next node's client thread; it forms a connection, which is passed to a new RequestHandler that sends the server's dht; the client compares the dht to its own and the hashed file contents to determine which files it wants from the server; it then requests the files and receives them from requesthandler

DHT

- Keys are fileNames
    - content is the hashed contents
- when two nodes connect, the client and server make new hashtables of their files
- the client compares its hashtable to the servers and requests any files that the server has that the client does not or which have different contents (see appendix figure 2)
    - we default to taking the server's copy of files if they are different

Michael Jee
Minh Truong
5/1/21
CECS 327

Improvements with more time

The network

- we could debug the network so that when the previous and next nodes of a node are the same node, there is no weird socket error
    - perhaps due to us not having main start the server and client threads correctly – that is, they are desync'd
- conversely we could implement Chord the proper way with the finger table and math and that would improve performance (log(n) for sending messages, updating, and storage)
- we could implement proper waiting to make some threads block until a connection is made instead of relying on Thread.sleep() to force waiting
- in findIpAddresses, instead of spawning new threads to ping each possible ip address in our subnet, we could make something to properly query an arp table to find nodes for our network
- we would implement a way to handle a node leaving the network and leaving abruptly
- we would implement a menu to join/leave the network and to sync files to handle errors from having searching for/joining a network and syncing files in the same 'thread'

DHT

- we would add in a timestamp and cache a hashed copy of each file to manage merge conflicts better
- on hash table conflict, we could look at each file conflict and compare each file's hashed contents to the cached copies and if they exist, we simply take the newest one and update the old copy, otherwise we cache this new copy and tell the user they need to manage the merge
    - we could organize the hashed value so that we could determine which file versions are parents of which children and use this information to highlight which parts of the file are different to users (like git)

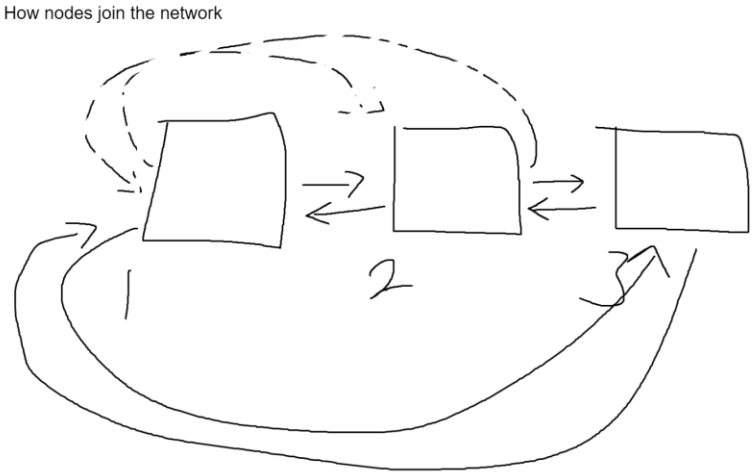
Michael Jee
Minh Truong
5/1/21
CECS 327

Output (works in reverse too, but depends on which order we start the processes)

Michael Jee
Minh Truong
5/1/21
CECS 327

Appendix

Figure 1



when no machines in a lan have our specific port open, there is no network
in this case, a node, 1 will create its own network and have "" as its next and previous 'pointers'

when there is one machine, node 1, in a network and node 2 tries to join, node 2 will set its previous and next to equal node 1's ip address and node 1 will set its previous and next as node 2's

when the network is 2 or more, node 3, the last node will always set its previous to the node it contacts (node 2) and its next to the contacted node's next; node 2 will keep its previous, set its next as node 3 and tell its old next (node 1) that its new previous is node 3

Figure 2