Project 4

Yao Liu
Concurrent nodes join

• In Project 3, nodes join one by one.
  • A join() RPC call won’t return until all fingertable updates, file transfers have taken place.
  • If implemented correctly, all nodes will have the correct fingertable entries.

• In Project 4, nodes may join the DHT concurrently.
  • We cannot guarantee nodes’ fingertables are all correct at all times.
  • but all fingertable entries will be updated over time and eventually set to their correct values
join()

• The join() function will be different from Project 3.
• It does not try to create a fingertable with all entries set.
  • fingertable entries (except the first one) are updated by the fixFinger() function.
• Instead, it only uses an existing DHT node to find its successor, i.e., the first entry in the fingertable
• notify() its successor to update the predecessor
getNodeSucc() and getNodePred()

- getNodeSucc() returns the node’s successor, which is the first entry in the node’s fingertable.

- getNodePred() returns the node’s predecessor
  - Initially, the node does not know about its predecessor, and this may return NULL (or a structure indicating the predecessor has not been set).

- getNodeSucc() and getNodePred() may not return the correct value, but the successor and predecessor are periodically updated by the stabilize() and notify() functions.
stabilize()

• A node \( n \) periodically checks if \( \text{pred}(\text{succ}(n)) \) belongs to the interval: \((n, \text{succ}(n))\)
  • \( \text{succ}(n) \) is the first entry in \( n \)’s fingertable
  • \( \text{pred}(m) \) returns node \( m \)’s predecessor by calling \( \text{getNodePred()} \) on node \( m \).
• If yes, there exists a node \( p \) between \( n \) and \( \text{succ}(n) \)
  • Set \( p \) to its own successor (set \( p \) to the first entry in its fingertable)
  • Use the \text{notify()} \) function to notify \( p \) that it (node \( n \)) is its predecessor
• If the predecessor of \( \text{succ}(n) \) is not set
  • \text{notify()} \( \text{succ}(n) \) that node \( n \) is its predecessor
notify()

- notify() takes one node ID, \( q \), as input argument
- Node \( n \) that receives the notify() RPC call
  - will set \( q \) as its predecessor if its predecessor is currently not set.
  - will set \( q \) as its predecessor if \( q \) is in the interval of (\( n \)’s current predecessor, \( n \))
stabilize() and notify() are called periodically and eventually, all predecessors and successors of all nodes in the DHT are eventually set to their correct values.
Use fixFingers() to update fingertable entries

- stabilize() and notify() correct nodes’ predecessors and successors
- Nodes’ fingertable entries are corrected by fixFingers()
- A node periodically selects a fingertable entry at random and updates it using findSucc(id).
Test your implementation

• You will need to write a test program to check if after a sufficient amount of time
  • all nodes’ successors are correct
  • all nodes’ predecessors are correct
  • the number of correct fingertable entries increases
  • the number of incorrect fingertable entries set by fixFingers first increases then decreases (unset fingertable entries are not counted as incorrect)

• Use cmp_fingertables to determine if fingertable entries are correct
The number of correct successors

64 DHT nodes in total
The number of correct/incorrect fingertable entries

64 DHT nodes in total
Extra 30% credit

• Extend your test program to gauge the performance of the stable Chord protocol.
• Need to output two additional pieces of information:
  • the average number of correct successor nodes returned by findSucc().
  • the average number of calls to findPred() for each correct successor returned
    • Use the optional count field in the NodeID structure.
64 DHT nodes in total