Peer-to-Peer

Apr 27 2016:

Recap:

- Advantages of p2p
    - p2p protocols allow users to download from one another instead of having a dedicated machine
    - Allows files to persist since there is no single point of failure

- **A p2p system needs to achieve two objectives:**
    - Search for the node who has the file
    - Download the file

  The search problem:
    - Naive: Napster, Single central server, single point of failure, bad load-balancing
    - Slightly better: Gnutella
        - Each peer knows about a few other peers
        - When a new query comes in, manufacture a new Query ID and flood to peers, who flood to their peers and so on.
            - (except incoming port, Like learning bridges)
        - What else should the request message contain? TTL values to prevent messages from going on forever
        - Once a node has the answer to a request, what must this node do?
            - Previously, sent response back through its neighbors. Long and inefficient
            - These days, just put the source IP and port inside the query ID, so that node can respond to the query-sender directly
        - How to bootstrap this system?
            - At the end of a session, keep track of who you connected to previously
            - Ship with a few hard-coded peers that are ‘always up’
        - Problems
            - Scale
            - Easy to download popular files, hard to find needles in haystacks
    - Enter DHTs

DHTs:

- Distributed Hash Tables are a solution to the search problem
- Basic idea is that you take a file and map it onto a key from a keyspace. For example, suppose you have a keyspace of 160 bits, then you take the filename, get the SHA-1

hash (k) and that'll be 160 bits long.

- Let's make it simpler. Suppose that we're going to each of you a separate final-exam. Instead of hosting it just on a single machine, we'll make it p2p.
- Each of you can potentially have numbers from 0 to 2^6. But we're still coming up with the exams, so the files aren't all there yet.
    - Example 'Chord'
    - Map 64 keys onto a circular space
    - Map existing node IDs also onto the same circular space
    - Key is mapped to the first node ID >= key. Achieves load-balancing since keys are assumed to be reasonably randomly distributed (We'll come back to this)
    - How do you search?
        - The easy way: Just maintain a mapping of who the successor is!
            - Efficiency: O(logN) where N is number of nodes
        - The efficient way: Maintain 'finger table'
            - For node 'n', 'i'th finger table entry is the first node whose ID >= n + 2 ^ i
    - For more details and for the figures that I used, check the Chord paper:
        - http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.125.663&rep=rep1&type=pdf
    - Abstracted the 'consistent hashing' part of mapping keys uniformly to keyspace. In reality it is done using 'consistent hashing' which is different from ordinary hashing.
- BitTorrent uses DHTs to find peers. Now, let's talk about BitTorrent

BitTorrent

- A few problems with previous p2ps
    - If everybody wants the same file at the same time the seeder has to upload lots of times
    - There exist freeloaders (they download but don't upload)
- What's the solution?
    - Upload *while* you download!
    - How? By breaking file into lots of pieces. Download pieces at a time and upload them
    - Randomize order of downloading pieces
    - Bootstrapping problem. What's the easiest solution?
        - There exists a tracker which keeps track of who the current peers are. The tracker lets peers know how to find each other
            - Problem? Tracker can be sued
        - Or use a DHT. A peer uses the hash of the torrent to find the ID of the torrent. Then it asks peers in its routing table where it can find that node ID. They forward it. Until the request eventually gets to a node who has

the peer list
    - The node then inserts itself onto the peer list for that node
- Free loader problem? Try to incentivize peers to share using ‘tit for tat’ and optimistic unchoking.
- Peers share pieces with other peers who have previously shared pieces (tit for tat)
- To allow new peers to gain a few pieces for free at the beginning, ‘optimistically unchoke’ a peer every now and then. That is, share a piece ‘for free’ to a random peer.
    - Also allows the peer to find other peers who may be faster than its current peers