Updates

• A3 due this Sunday
• Don’t forget to Piazza vote for your “worst weeks”
416 Distributed Systems

Feb 2, Peer-to-Peer
Outline

• P2P Lookup Overview
• Centralized/Flooded Lookups
• BitTorrent
• Routed Lookups – Chord
Scaling Problem

- Millions of clients $\Rightarrow$ server and network meltdown
P2P System

- Leverage the resources of client machines (peers)
  - Traditional: Computation, storage, bandwidth
  - Non-traditional: Geographical diversity, mobility, special token we call coins, sensors!
Peer-to-Peer (storage) Networks

• Typically each member stores/provides access to content
• Basically a replication system for files
  • Always a tradeoff between possible location of files and searching difficulty
  • Peer-to-peer allow files to be anywhere → searching is the challenge
  • Dynamic member list makes it more difficult: node churn
• What other systems have similar goals?
  • Routing, CDNs, DNS
The Lookup Problem

Key=“title”
Value=MP3 data...

Publisher

Client

Lookup(“title”)
Searching

• Needles vs. Haystacks
  • Searching for top 40, or an obscure punk track from 1981 that nobody’s heard of?

• Search expressiveness
  • Whole word? Regular expressions? File names? Attributes? Whole-text search?

• Searching for recent versus older content
• Searching for content correlated with your location/time of day/etc versus not
Framework

• Common Primitives:
  • **Join**: how do I begin participating?
  • **Publish**: how do I advertise my file?
  • **Search**: how to I find a file?
  • **Fetch**: how to I retrieve a file?
Outline

- P2P Lookup Overview
- Centralized/Flooded Lookups
- BitTorrent
- Routed Lookups – Chord
Napster: Overview

• Centralized Database:
  - **Join**: on startup, client contacts central server
  - **Publish**: reports list of files to central server
  - **Search**: query the server => return someone that stores the requested file
  - **Fetch**: get the file directly from peer
Napster: Publish

I have X, Y, and Z!

123.2.21.23

insert(X, 123.2.21.23)
Napster: Search

Where is file A?

123.2.0.18

search(A) --> 123.2.0.18

Fetch

Query

Reply
Napster: Discussion

- **Pros:**
  - Simple
  - Search scope is $O(1)$
  - Controllable (pro or con?)

- **Cons:**
  - Server maintains $O(N)$ State
  - Server does all processing
  - Single point of failure
Napster: Discussion

- **Pros:**
  - Simple
  - Search scope is $O(1)$
  - Controllable (pro or con?)

- **Cons:**
  - Server maintains $O(N)$ State
  - Server does all processing
  - **Single point of failure**
“Old” Gnutella: Overview

• **Query Flooding:**
  • **Join:** on startup, client contacts *a few* other nodes; these become its “neighbors”
    • “unstructured overlay”
  • **Publish:** no need
  • **Search:** ask neighbors, who ask their neighbors, and so on... when/if found, reply to sender.
    • TTL limits propagation
  • **Fetch:** get the file directly from peer
Gnutella: Search

I have file A.

I have file A.

Where is file A?

Query

Reply

I have file A.

17
Gnutella: Discussion

- **Pros:**
  - Fully de-centralized
  - Search cost distributed
  - Processing @ each node permits powerful search semantics

- **Cons:**
  - Search scope is $O(N)$
  - Search time is $O(???)$
  - Nodes leave often, network unstable

- **TTL-limited search works well for haystacks.**
  - For scalability, does NOT search every node. May have to re-issue query later; no guarantee that it will find the file!
• Modifies the Gnutella protocol into two-level hierarchy
  • Hybrid of Gnutella and Napster

• Supernodes
  • Nodes that have better connection to Internet
  • Act as temporary indexing servers for other nodes
  • Help improve the stability of the network

• Standard nodes
  • Connect to supernodes and report list of files
  • Allows slower nodes to participate

• Search
  • Broadcast (Gnutella-style) search across supernodes

• Disadvantages
  • Kept a centralized registration → allowed for law suits 😞
Outline

• P2P Lookup Overview

• Centralized/Flooded Lookups

• BitTorrent

• Routed Lookups – Chord
BitTorrent: Overview

- **File swarming:**
  - **Join:** contact centralized “tracker” server, get a list of peers.
  - **Publish:** Run a tracker server.
  - **Search:** Out-of-band. E.g., use Google to find a tracker for the file you want.
  - **Fetch:** Download chunks of the file from your peers. Upload chunks you have to them.

- **Big differences from Napster:**
  - Chunk based downloading
  - “few large files” focus
  - Anti-freeloading mechanisms
  - Out-of-band with search engines scalable and resilient
BitTorrent: Publish/Join

Seeder

Tracker
BitTorrent: Fetch

Seeder
BitTorrent: Sharing Strategy

- Employ “Tit-for-tat” sharing strategy
  - A is downloading from some other people
    - A will let the fastest N of those download from it
  - Be optimistic: occasionally let freeloaders download
    - Optimistic unchoke
    - Otherwise no one would ever start!
    - Also allows you to discover better peers to download from when they reciprocate
  - Rarest first policy: distribute rare blocks first

- Goal: Pareto Efficiency
  - Game Theory: “No change can make anyone better off without making others worse off”
  - Does it work? How would you cheat?
  - (not perfectly, but perhaps good enough?)
BitTorrent: Summary

- **Pros:**
  - Works reasonably well in practice
  - Gives peers incentive to share resources; avoids freeloaders

- **Cons:**
  - Pareto Efficiency claim is not true ... a lie
  - Central tracker server needed to bootstrap swarm
    - Alternate tracker designs exist (e.g., DHT-based trackers)
Outline

• P2P Lookup Overview

• Centralized/Flooded Lookups

• BitTorent

• Routed Lookups (DHTs) – Chord (another example: Kademlia)
The Lookup Problem

Publisher

Key="title"
Value=MP3 data...

Client

Lookup("title")
DHT: Overview (1)

- Goal: make sure that an item (file) identified is always found in a reasonable # of steps
- Abstraction: a distributed hash-table (DHT) data structure
  - insert(id, item);
  - item = query(id);
  - Note: item can be anything: a data object, document, file, pointer to a file…
- Implementation: nodes in system form a distributed data structure
  - Can be Ring, Tree, Hypercube, Skip List, Butterfly Network, ...
DHT: Overview (2)

- **Structured Overlay Routing:**
  - Usually builds on *consistent hashing*:
    - *Items and nodes are hashed into the same ID space*
  - **Join:** On startup, contact a “bootstrap” node and integrate yourself into the distributed data structure; get a *node id*
  - **Publish:** Route publication for *file id* toward a close *node id* along the data structure
  - **Search:** Route a query for file id toward a close node id. Data structure guarantees that query will meet the publication.
  - **Fetch:** Two options:
    - Publication contains actual file => fetch from where query stops
    - (Indirection) Publication says “I have file X” => query tells you 128.2.1.3 has X, use IP routing to get X from 128.2.1.3
DHT: Example - Chord

- Associate to each node and file a unique *id* in an *uni*-dimensional space (a Ring)
  - E.g., pick from the range \([0...2^m]\)
  - Usually the hash of the file or IP address

- Routing properties:
  - Routing table size is \(O(\log N)\) , where \(N\) is the total number of nodes
  - Guarantees that a file is found in \(O(\log N)\) hops

From MIT in 2001
A key is stored at its **successor**: node with next higher ID.
Routing: Chord Basic Lookup

"Where is key 80?"

"N90 has K80"
Chord: finger tables (fast lookup)

- Assume identifier space is $0 \ldots 2^m$
- Each node maintains
  - Finger table
    - Entry $i$ in the finger table of $n$ is the first node that succeeds or equals $n + 2^i$
  - Predecessor node
- An item identified by $id$ is stored on the successor node of $id$
Routing: Chord Example

- Assume an identifier space 0..7
- Node n1: (1) joins → all entries in its finger table are initialized to itself
Routing: Chord Example

• Node n2(2) joins
Routing: Chord Example

- Nodes n3:(0), n4:(6) join
Routing: Chord Examples

- Nodes: n1(1), n2(2), n3(0), n4(6)
- Items: file1:(7), file2:(1)
Routing: Query

- Upon receiving a query for item \( id \), a node
  - Check whether stores the item locally
  - If not, forwards the query to the largest node in its successor table that does not exceed \( id \)
DHT: Chord Summary

- Routing table size?
  - $\log N$ fingers
- Routing time?
  - Each hop expects to $1/2$ the distance to the desired id $\Rightarrow$ expect $O(\log N)$ hops.
- Pros:
  - Guaranteed Lookup
  - $O(\log N)$ per node state and search scope
  - Influenced many future systems; esp. key-val stores
- Cons:
  - No one uses them? (BitTorrent somewhat)
  - Supporting non-exact match search is hard
What can DHTs do for us?

- Distributed object lookup
  - Based on object ID
- De-centralized file systems
  - CFS, PAST, Ivy
- Application Layer Multicast
  - Scribe, Bayeux, Splitstream
- Databases
  - PIER
When are p2p / DHTs useful?

- Caching and “soft-state” data
  - Works well! BitTorrent, KaZaA, etc., all use peers as caches for hot *read-only* data

- Finding read-only data
  - Limited flooding finds hay
  - DHTs find needles

- BUT
A Peer-to-peer ?

- Complex intersection queries ("the" + "who")
  - Billions of hits for each term alone
- Sophisticated ranking
  - Must compare many results before returning a subset to user
- Very, very hard for a DHT / p2p system
  - Need high inter-node bandwidth
  - (This is exactly what Google does - massive clusters)
Writable, persistent p2p

- Do you trust your data to 100,000 monkeys?
- Node availability hurts
  - Ex: Store 5 copies of data on different nodes
  - When someone goes away, you must replicate the data they held
  - Hard drives are *huge*, but edge network upload bandwidth is tiny
  - May take days to upload contents of a hard drive. P2P replication/fault-tolerance expensive.
P2P: Summary

- Many different styles; remember pros and cons of each
  - centralized, flooding, swarming, and structured routing
- Lessons learned:
  - Single points of failure are very bad
  - Flooding messages to everyone is bad
  - Underlying network topology is important
  - Not all nodes are equal
  - Need incentives to discourage freelading
  - Privacy and security are important
  - Structure can provide theoretical bounds and guarantees