Lecture 24

CS625: Advanced Computer Networks
Fall 2003

Tuesday, 14 October 2003

Bhaskaran Raman
CSE, IIT-Kanpur

http://www.cse.iitk.ac.in/users/braman/courses/cs625-fall2003/outline.html

Topic for Today

- Peer-to-Peer Overlay Networks
- *Scribe for today?*

Peer-to-Peer Networks

- *Peers* form a network among themselves
  - Overlay on top of IP
  - Provide routing: usually towards named objects
- Classification
  - Centralized directory (e.g. Napster)
  - Decentralized directory
    - Unstructured (e.g. Gnutella, FastTrack, Morpheus)
    - Structured: topic for today

Structured Peer-to-Peer Networks

- Close coupling between network topology/addressing and data location
- Several distributed data structures proposed:
  - Chord
  - CAN (Content Addressable Network)
  - Tapestry

Peer-to-Peer Network Features

- Redundant storage
- Selection of nearby servers
- Searching of data
- *Efficient location of data item*

Chord Functionality

- Chord maps given *key* to a *node*
  - That node stores and serves data
  - Key space is flat
  - Can support dynamic node join/leave
- Application is responsible for other functionalities:
  - User friendly naming of data
  - Authentication
  - Replication

Example Applications

- Distributed indexing: Gnutella or Napster like keyword search
- Cooperative mirroring
- Time-shared storage
- Large-scale combinatorial search

The Chord Interface

- Two main interfaces with application above:
  - **lookup(key)**
  - **Call back** when the set of keys the node is responsible for changes

The Chord Lookup Protocol

- Logical identifier space, circular with *m*-bits
- Nodes and keys are hashed onto this space using SHA-1
  - Nodes' IP addresses are hashed
  - Consistent hashing: load balancing
- Key *k* is assigned to *successor node* in identifier space
  - Reassignment happens when nodes join/leave

Scalable Key Location

- Default lookup method:
  - Maintain and follow *successor* pointers
  - Takes O(N) time for lookup
- Optimization:
  - Maintain *finger* table with at most *m* entries
  - The *i*th entry will have the pointer to the *first* node *s* which is away from *n* by at least *2^(i-1)*
    - *s = successor(n + 2^(i-1))*

How the Finger Table Works

- Each node stores only a small number of pointers
- But any single node does not necessarily have pointer to *successor(k)*
- Node *n* searches its finger table for node *j* which most immediately precedes *k*
- In each step, we move closer to *k*
  - We at least halve the distance to *k* each time
  - O(logN) lookup time

Handling Node Join

- Invariants to maintain
  - Each node's *successor* is correctly maintained
  - For every *k*, *successor(k)* is responsible for *k*
- Maintain *predecessor* pointer for simplification
- Main steps when *n* joins:
  - Learn of some other node *n'* offline
  - Initialize own finger table and predecessor
  - Update fingers of existing nodes
  - Transferring keys
- O(logN * logN) messages

Some Remarks

- Handling simultaneous join, or node failure is a little bit more involved
  - Need to store *r* nearest successors
  - Need to replicate data associated with keys
- Protocol can be implemented *iteratively* or *recursively*
- Path length expansion is an important concern

Further topics this week...

- Internet Security