

CIS 700/005

Networking Meets Databases

Boon Thau Loo
Spring 2007
Lecture 3

Note: Several slides are courtesy of CS286 (Spring '05) at UC Berkeley

Announcements

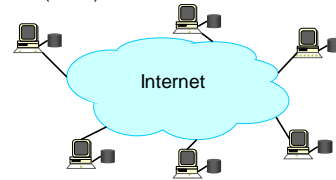
- Paper summaries due at noon today.
- Office hours: Wed 3-4 pm (605 Levine)
 - Project proposal: due **Feb 12**.
- Student presenter:
 - **23rd Jan**: "A Scalable Distributed Information Management System". SIGCOMM 2003.

Today

- Two assigned papers:
 - Chord (SIGCOMM '01)
 - Distributed Hash Tables (CACM '03)
- Focus on CAN and Chord

How Did it Start?

- A killer application: Napster
 - Free music over the Internet
- Key idea: share the *content*, storage *and* bandwidth of individual (home) users

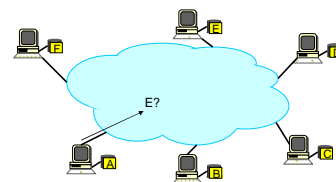


Model

- Each user stores a subset of files
- Each user has access (can download) files from all users in the system

Main Challenge

- Find where a particular file is stored



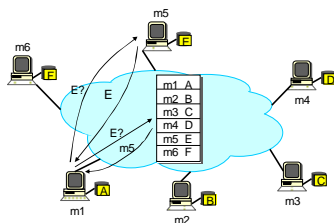
Other Challenges

- Scale: up to hundred of thousands or millions of machines
- Dynamicity: machines can come and go any time

Napster

- Assume a centralized index system that maps files (songs) to machines that are alive
- How to find a file (song)
 - Query the index system → return a machine that stores the required file
 - Ideally this is the closest/least-loaded machine
 - ftp the file
- Advantages:
 - Simplicity, easy to implement sophisticated search engines on top of the index system
- Disadvantages:
 - Robustness, scalability (?)

Napster: Example

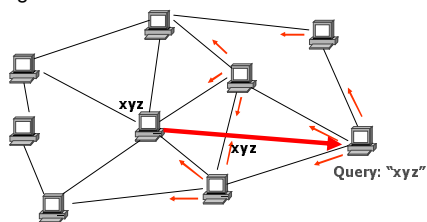


Gnutella

- Distribute file location
- Idea: flood the request
- How to find a file:
 - Send request to all neighbors
 - Neighbors recursively multicast the request
 - Eventually a machine that has the file receives the request, and it sends back the answer
- Advantages:
 - Totally decentralized, highly robust
- Disadvantages:
 - Not scalable; the entire network can be swamped with request (to alleviate this problem, each request has a TTL)

Gnutella

- Ad-hoc topology
- Queries are flooded for bounded number of hops
- No guarantees on recall



Distributed Hash Tables (DHT's)

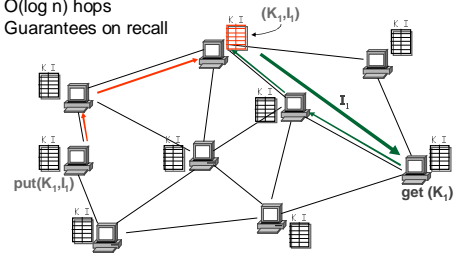
- Abstraction: a distributed hash-table data structure
 - put(key, item);
 - item = get(key)
 - Note: item can be anything: a data object, document, file, pointer to a file...
- Proposals
 - CAN, Chord, Pastry, Tapestry, etc

DHT Design Goals

- Make sure that an item (file) identified is always found
- Scales to hundreds of thousands of nodes
- Handles rapid arrival and failure of nodes

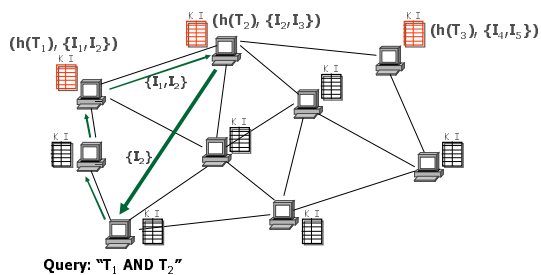
Structured Networks (aka DHTs)

- Distributed Hash Tables (DHTs)
- Hash table interface: `put(key,item)`, `get(key)`
- $O(\log n)$ hops
- Guarantees on recall



Keyword Search using DHTs

- *Inverted Lists* hashed by keyword (term) in the DHT



Usage of DHTs

- **Rendezvous:**
 - The key (or identifier) is globally unique and static
 - DHT functions as a dynamic and flat DNS
 - E.g. IP mobility, Chat, Internet telephony, DNS, The Web!
- **Storage:**
 - Rendezvous applications use the DHT only to store small pointers (IP addresses, etc.)
 - DHTs for more serious storage, such as file systems
 - Chord File System (SOSP 2001)

Usage of DHTs

- **Internet-scale Query Processing**
 - Use the DHT as a global hash index
 - Distributed joins and aggregations using DHTs
 - **PIER (VLDB '03, CIDR '05)**
- **Others:**
 - Multicast, publish-subscribe, event-notification, etc
- **Data-centric Internet Architecture**
 - **i3 (SIGCOMM '03), ROFL (SIGCOMM '06)**

DHT-based Research Projects

- File sharing [CFS, OceanStore, PAST, ...]
- Web cache [Squirrel, CoralCDN]
- Backup store [Pastiche]
- Database Query Engines [PIER, ...]
- Event notification [Scribe]
- Naming systems [ChordDNS, Twine, ...]
- Communication primitives [i3, ROFL]
- Multimedia streaming [SplitStream]
- Sensor networks [GHT – DHT + GPSR]

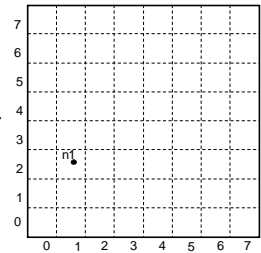
Common thread: data is location-independent

Content Addressable Network (CAN)

- Associate to each node and item a unique id in an d -dimensional Co-ordinate space
- Properties
 - Routing table size $O(d)$
 - Guarantees that a file is found in at most $d * n^{1/d}$ steps, where n is the total number of nodes

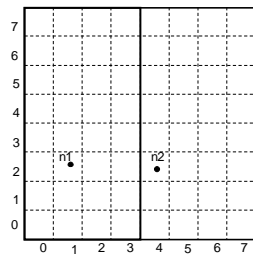
CAN Example: Two Dimensional Space

- Space divided between nodes
- All nodes cover the entire space
- Each node covers either a square or a rectangular area of ratios 1:2 or 2:1
- Example:
 - Node $n_1:(1, 2)$ first node that joins \rightarrow cover the entire space



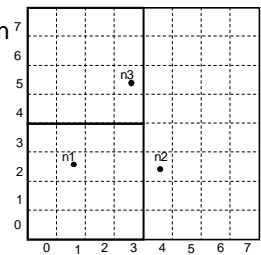
CAN Example: Two Dimensional Space

- Node $n_2:(4, 2)$ joins \rightarrow space is divided between n_1 and n_2



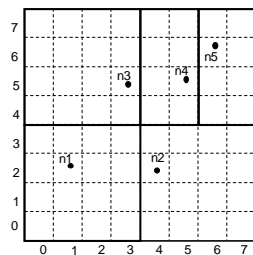
CAN Example: Two Dimensional Space

- Node $n_2:(4, 2)$ joins \rightarrow space is divided between n_1 and n_2



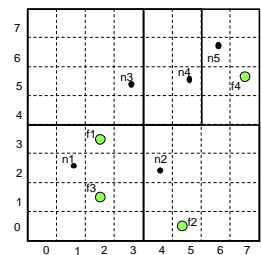
CAN Example: Two Dimensional Space

- Nodes $n_4:(5, 5)$ and $n_5:(6, 6)$ join



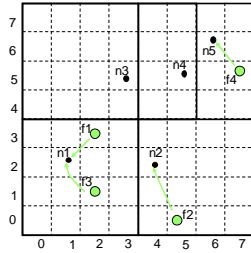
CAN Example: Two Dimensional Space

- Nodes: $n_1:(1, 2)$; $n_2:(4, 2)$; $n_3:(3, 5)$; $n_4:(5, 5)$; $n_5:(6, 6)$
- Items: $f_1:(2, 3)$; $f_2:(5, 1)$; $f_3:(2, 1)$; $f_4:(7, 5)$



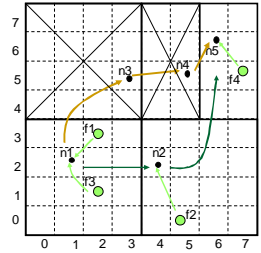
CAN Example: Two Dimensional Space

- Each item is stored by the node who owns its mapping in the space

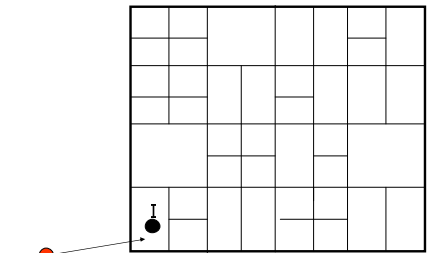


CAN: Query Example

- Each node knows its neighbors in the d -space
- Forward query to the neighbor that is closest to the query id
- Example: assume $n1$ queries $f4$
- Can route around some failures

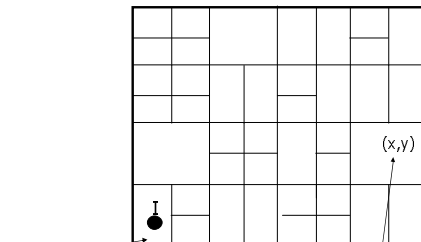


CAN: Node Joining



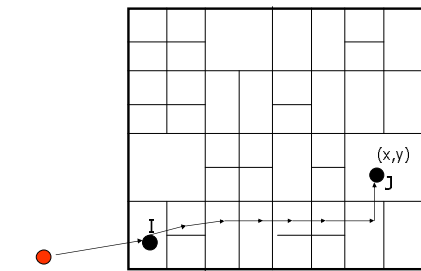
new node 1) Discover some node "I" already in CAN

CAN: Node Joining



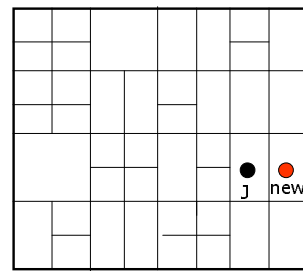
new node 2) Pick random point in space

CAN: Node Joining



new node 3) I routes to (x,y) , discovers node J

CAN: Node Joining



4) split J's zone in half... new node owns one half

Node departure

- Node explicitly hands over its zone and the associated (key,value) database to one of its neighbors
- In case of network failure this is handled by a take-over algorithm
- Problem : take over mechanism does not provide regeneration of data
- Solution: every node has a backup of its neighbours

Chord

- Associate to each node and item a unique *id* in an *uni-dimensional space* $0..2^m-1$
- Key design decision
 - Decouple correctness from efficiency
- Properties
 - Routing table size $O(\log(N))$, where N is the total number of nodes
 - Guarantees that a file is found in $O(\log(N))$ steps

Review: Simple Hashing

- Given document XYZ, we need to choose a server to use
- Suppose we use modulo
- Number servers from $1..n$
 - Place document XYZ on server $(XYZ \bmod n)$
 - What happens when a servers fails? $n \rightarrow n-1$
 - Why might this be bad?

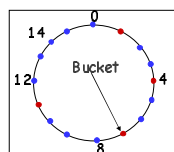
Consistent Hash [Karger 97]

- David Karger, et al. **Consistent Hashing and Random Trees: Tools for Relieving Hot Spots on the World Wide Web.** STOC '97.
- Desired features
 - Balanced – load is equal across buckets
 - Smoothness – little impact on hash bucket contents when buckets are added/removed
 - Spread – small set of hash buckets that may hold a set of object
 - Load – # of objects assigned to hash bucket is small

Consistent Hash [Karger 97]

Construction

- Assign each of C hash buckets to random points on mod 2^n circle, where, hash key size = n .
- Map object to random position on circle
- Hash of object = closest clockwise bucket

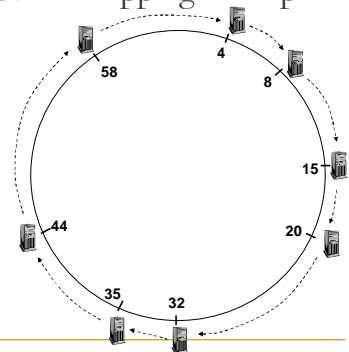


- Smoothness → addition of bucket does not cause movement between existing buckets
- Spread & Load → small set of buckets that lie near object
- Balance → no bucket is responsible for large number of objects

Identifier to Node Mapping Example

- Node 8 maps [5,8]
- Node 15 maps [9,15]
- Node 20 maps [16, 20]
- ...
- Node 4 maps [59, 4]

- Each node maintains a pointer to its successor



Lookup

- Each node maintains its successor
- Route packet (ID, data) to the node responsible for ID using successor pointers

Joining Operation

- Node with id=50 joins the ring via node 15

Joining Operation

- Node 50: send join(50) to node 15
- Node 44: returns node 58
- Node 50 updates its successor to 58

Joining Operation

- Node 50: send stabilize() to node 58
- Node 58:
 - update predecessor to 50
 - send notify() back

Joining Operation (cont'd)

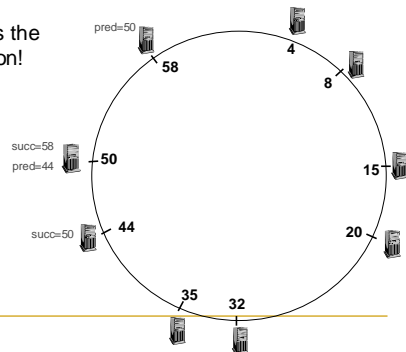
- Node 44 sends a stabilize message to its successor, node 58
- Node 58 reply with a notify message
- Node 44 updates its successor to 50

Joining Operation (cont'd)

- Node 44 sends a stabilize message to its new successor, node 50
- Node 50 sets its predecessor to node 44

Joining Operation (cont'd)

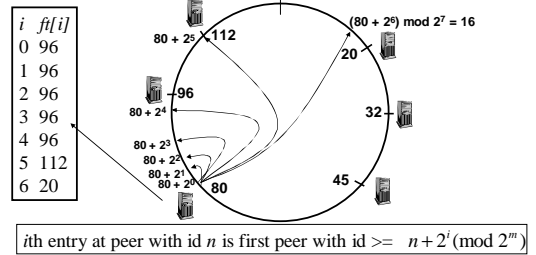
- This completes the joining operation!



Achieving Efficiency: *finger tables*

Finger Table at 80

Say $m=7$



CAN/Chord Optimizations

- Reduce latency
 - Chose finger that reduces expected time to reach destination ("*route selection*")
 - Chose the closest node from range $[N+2^{i-1}, N+2^i)$ as i th finger ("*proximity neighbor selection*")
- Accommodate heterogeneous systems
 - Multiple virtual nodes per physical node

Achieving Robustness

- To improve robustness each node maintains the k (> 1) immediate successors instead of only one successor
- In the notify() message, node A can send its $k-1$ successors to its predecessor B
- Upon receiving notify() message, B can update its successor list by concatenating the successor list received from A with A itself

Good papers to follow-up

- Krishna Gummadi et al, **The Impact of DHT Routing Geometry on Resilience and Proximity**, SIGCOMM'03
- Sean Rhea, et al. **Handling Churn in a DHT**. *Proceedings of the USENIX Annual Technical Conference*, June 2004.
- Frank Dabek, et al. **Designing a DHT for Low Latency and High Throughput**, NSDI 2004.
- Castro, et al. **Secure Routing for Structured Peer-to-Peer Overlay Networks**, OSDI 2002

Geometries we compare

Geometry	Algorithm
Ring	Chord, Symphony
Hypercube	CAN
Tree	Plaxton
Hybrid = Tree + Ring	Tapestry, Pastry

Summary of our flexibility analysis

Flexibility	Ordering of Geometries
Neighbors (FNS)	Hypercube << Tree, XOR, Ring, Hybrid (1) (2 ⁱ⁻¹)
Routes (FRS)	Tree << XOR, Hybrid < Hypercube < Ring (1) (logN/2) (logN/2) (logN)

Available Software

- Chord: <http://pdos.csail.mit.edu/chord/> (C++)
- FreePastry: <http://freepastry.org/> (Java)
- Bamboo: <http://bamboo-dht.org/> (Java)
- Open DHT: <http://opendht.org/> (Java/Per/Python)
- p2psim: <http://pdos.csail.mit.edu/p2psim/>

Next lecture

- This Thursday:
 - **Querying the Internet with PIER** (VLDB '03)
 - **A Knowledge Plane for the Internet** (SIGCOMM '03)
- Summaries due at noon on Thursday.