CS345: Lecture 2

Introduction to DHTs

Recap

• Three major characteristics of P2P
  – Dynamism
  – Scale (Number of Nodes)
  – Physical Distribution (Internet-wide)
• Challenges arise from all three, individually and in combination

Two Models for P2P Systems

1. A distributed/federated database on steroids
   – Each node owns some content
   – E.g., File-sharing systems
2. A parallel database on steroids
   • P2P as a service: store client data, answer queries
   • (1) may use (2) as sub-component

A Simple Parallel P2P System

• Provide a Dictionary ADT
  – Data Update: Ins(key,value), Del(key,value)
  – Queries: LookUp(key)
• Operations initiated at any node
• Centralized, in-memory data structure?
  – Binary search trees, Hash table
  – Why hashing?

A Popular Solution

• Issues in a P2P system
  – Load balance
  – Dynamism: Join(node), Leave(node)
  – Scale: Need sub-linear operations
  – Network: Low latency
• Popular solution: Distributed Hash Tables (DHTs)
  – Chord, CAN, Pastry, Tapestry, Symphony, Kademlia,
    P-grid, Viceroy, Koord, LAND, …

Distributing a Hash Table

• Part 1: Storage
  – Who stores what? What happens with dynamism?
  – How cheap are the operations?
  – What about load balance?
• Part 2: Routing
  – Send query/insert/delete to the right node
  – Cost of dynamism?
  – Latency of routing? Messages/Bandwidth needed?
The Storage Problem

• Hash function $h: \{\text{keys}\} \rightarrow \{0, 2^N\}$
  – Circular N-bit hash space
• Each node has N-bit ID
  – Node $N_i$ maintains segment $[N_i, N_{i+1})$
  – ID chosen uniformly at random (for now)

Dealing with Dynamism

• Node join
  – Pick a random ID
  – Take over appropriate segment
  – Cost: Data transfer
• Node leaves
  – Detect event
  – Predecessor expands range
  – What about data loss?

Handling Data loss

• Replicate at $r$ predecessors/successors
  – How to make updates/queries ACID?
  – Ideas from CS347…
• Soft-state approach
  – Expect owners to refresh data periodically
  – “best-effort” service, no availability guarantee
• Question: Why is hashing useful?

Load Balance

• Hashing distributes data uniformly
  – Under what assumptions?
• Random node IDs $\Rightarrow$ Equal-size segments
  – Really? Stay tuned…
• Equal number of tuples on all nodes
  – Even with dynamism
  – Is this load balance?

The Routing Problem

• So far…
  – Who stores what (in a dynamic network)?
  – How is load balance achieved?
• Now: How to send query/update to the right guy?
  – Data and node operations must be efficient (cost = \# messages)
  – Also deal with physical network issues (cost = n/w bandwidth, latency)

A Naïve Solution

• Maintain directory $\langle\text{node ID, IP address}\rangle$
  – Routing cost is $O(1)$
• Centralized. What is the problem?
  – Not P2P.
  – Directory is single point of failure.
• Replicate at all nodes.
  – Cost of node join/leave is $\Theta(n)$
An Alternative Solution

- Use a linked list
  - Pass messages around until destination reached
- Maintaining the list
  - Connect to multiple successors (braided list)
  - Cost of maintenance $O(1)$
- Cost of query?
  - $\theta(n)$ messages for routing

Identifying the trade-off

- Two overlay networks
  - Complete graph: fast routing, high maintenance
  - Linked list: slow routing, low maintenance
- Something in between?
  - Lots!
  - Adaptations of classical parallel interconnection networks, e.g., hypercube, butterfly.

Chord

- Add skip pointers to the linked list
- Node $x$ links to $\text{succ}(x + 2^i)$, for all $i > 0$

Chord (2)

- How many links per node?
  - Expected value roughly $\log n$
  - Why? IDs chosen at random
- How are links used?
  - Greedy routing to destination!

Chord (3)

- How many hops for routing?
  - $O(\log n)$ (Roughly $0.5 \log n$ in expectation)
  - Distance reduces by about half in each step
  - No nodes within a distance $2^i/n$
- What is maintenance cost?
  - Links change as nodes join and leave
  - $O(\log n)$ messages per join/leave when optimized

Node Join

1. Bootstrap: Need one other node in system
2. Query it for your own ID.
3. Talk to query destination to join ring.
4. Establish your links (use predecessor’s links as hint).
5. Successor corrects incorrect incoming links.
Node Leaves and Consistency

- When a node leaves
  - Neighbor detects through heartbeats
  - Fix links like in node joins
- Consistency of braided list is critical
  - Other pointers are an optimization
  - Need careful serialization for concurrent joins

Other DHT structures: Pastry

- Different set of skip pointers: hypercubic structure
  - For each i, link to any node with common i-bit prefix (and different (i+1)th bit)
  - Node 100110010 links to 0*
    11*
    101*
    1000*
    .......
  - Non-deterministic

Routing in Pastry

- Prefix routing, followed by braided list
  - Left-to-right bit fixing
- Example
  - Src = 100110010 Dest = 101101101
  - Use third link from Src to 101*
  - When no such link exists, go along linked list

Routing Properties and Changing Bases

- Similar to Chord
  - O(log n) links per node, O(log n) routing hops
- Is it possible to obtain different trade-offs?
  - Construct in higher bases instead of base 2
  - More links per node, fewer hops
- Is the trade-off optimal?
  - No. More later…
**DHTs and Small Worlds: A Digression**

- Chord resembles a social network
  - Braids reflect social clusters
  - Skip pointers are *weak ties*. Critical for routing.
- Kleinberg’s Small World construction
  - A model of social networks
  - Highly clustered graph
  - Yet supports efficient *greedy* routing

**Kleinberg’s Construction**

- Ring of nodes
  - *a la* Chord
- Each node $m$ has one long-distance link
  - Not uniformly random
  - $P(m \text{ choosing } x) = 1/\text{dist}(m,x)$
  - More likely to know someone close to me
- Simple *Greedy* routing like in Chord
  - works in $\theta(\log^2 n)$ hops!

**The Shape of the Distribution**

![Distribution Graph]

**Extending Kleinberg’s Construction: Symphony**

- What if nodes have more long-distance links?
  - Choose $k$ links independently like in Kleinberg
  - Greedy routing in $O(\log^2 nk)$
  - When $k=\log n$, greedy routing in $O(\log n)$
- What does the structure look like?
  - Randomized version of Chord
  - Links are exponentially distributed in distance

**More Routing Structures**

- Chord/Pastry are *sub-optimal*
  - Possible to route in fewer hops for same maintenance cost
- Different structures offer optimal trade-off
  - $O(\log n)$ links, $O(\log n / \log \log n)$ hops
  - Viceroy uses butterflies
  - Koore : de Bruijn graphs
  - Randomized Topologies: Symphony with lookahead routing

**Optimizing for the Physical Network**

- Minimizing # hops isn’t good enough
  - What if hop is from Stanford to Timbuktu?
  - More important: query routing latency
- Can we make sure most hops are “short”?
  - Minimize *stretch*: ratio of routing latency to actual *src-dest* internet latency
Some Intuition

- Many different approaches possible
  - Smart overlay embedding [GSM03], smart construction [Pastry], smart routing [CAN]
  - Focus on smart construction
- Intuition: The power of multiple choices
  - Closest of $k$ randomly chosen nodes is very close on the Internet
  - $k=16-32$ close to optimal latency
  - True of all power-law expansion graphs

Exploiting the Observation

- Solution: Introduce choices for each link a node sets up
  - Different options for this too!
- Recall Chord rule: $x$ links to $\text{succ}(x+2^i)$
  - Relax to allow any node in $(x+2^i, x+2^{i+1})$
  - Sample $k$ nodes and connect to closest
  - Nice properties about $k$ hops still hold
- Pastry: already non-deterministic!
- Symphony: Different solution needed [MGK04]

Internet Latency vs. Number of Samples: Data from the Skitter project

Looking Ahead

- Improving and Extending DHT designs
  - Caching and load balance
  - Support for sub-groups, access control, hierarchies [next week]
- Using DHTs in applications
  - Distributed object location and storage
  - Cooperative caches, DNS
  - Many others….

Looking Ahead (2) : Going beyond the Dictionary ADT

- Supporting Information Retrieval (IR) queries
  - Potentially use DHT as solution component
- Supporting relational queries
  - Range queries, Joins
  - May require new storage and routing designs
- Understanding and Guaranteeing Semantics
  - Complications from dynamism
  - Transactional semantics

Looking Ahead (3) : Security and Incentives

- Open P2P systems introduce complications
  - How to stop malicious users?
  - How to eliminate free-riders?