Distributed Indexing

Arvind Krishnamurthy
Fall 2004

Motivation

- Distributed Indexing generalizes in two ways:
  - System stores objects with multiple attributes
  - Objects can be retrieved by using one or many of the attributes
  - Should be able to support complex operations such as:
    - Nearest-neighbor queries
    - Range queries

- Motivating applications:
  - Large number of documents
  - Networking probing information
  - Sensor networks gathering data

Approaches

- Three current solutions:
  - PIER system (no range queries or nearest-neighbor queries)
  - pSearch system
  - Mercury system
  - SkipIndex system

- PIER:
  - Uses a hybrid of “unstructured” and “structured” networks
  - Uses the “unstructured” network without guarantees
  - Uses the “structured” network to guarantee retrieval

Unstructured Networks

- Ad-hoc topology
- Queries are flooded for bounded number of hops
- No guarantees on recall
- E.g. Gnutella and Kazaa

Structured Networks

- 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

Query: “xyz”
PIER Approach

Flood-based Network (All items) vs. DHT (Partial Index on Rare Items)

Simpler alternative to either optimizing unstructured or structured networks.

Flood Query vs. DHT Query

Very Few Results? vs. More Results

Unstructured Networks: Queries with Small Result Sets

Small result set → items have few copies in network

Query Latency

Queries that return few results have poor response times

DHT-based Search

Advantages:
- Avoid flooding query in network
- Guarantee recall (critical for small result sets)

Disadvantages:
- Hashing inverted lists into the DHT is costly
- So is intersecting inverted lists at query time
- Infeasible for Google-like datasets (IPTPS '03)

Feasible for querying rare items:
- Queries with ≤10 results ship 7x fewer inverted list entries compared to the average query
- Query optimization can reduce communication overhead: intersect rare terms first

Mercury: Supporting Range Queries

- Start with single-attribute data objects
  - Use DHT's without hash functions
  - Results in load-imbalance
- Address load-imbalance
  - Results in longer routing distances
  - Modify routing tables
- Generalize to multiple attributes

Using DHTs for Range Queries

- No cryptographic hashing for key → identifier

Query: 6 ≤ x ≤ 13
key = 6 → 0x0ab
key = 7 → 0x0cd
key = 12 → 0x012

Query: 6 ≤ x ≤ 13
key = 6 → 0x0ab
key = 7 → 0x0cd
key = 12 → 0x012
ARB1  difference between cryptographic hashing and not hashing -- animate... hash(x) = blah, hash(y) = blah_2

Ashwin, 8/25/2004
Using DHTs for Range Queries

- Nodes in popular regions can be overloaded
- Load imbalance!

DHTs with Load Balancing

- Mercury load balancing strategy
  - Re-adjust responsibilities
- Range ownerships are skewed!

DHTs with Load Balancing

- Each routing hop may not reduce node-space by half!
  - \( \rightarrow \) no \( \log(n) \) hop guarantee

Ideal Link Structure

- Need to establish links based on node-distance

Mercury's Solution

- Need to establish links based on node-distance
  - If we had the above information...
  - For finger \( i \)
    - Estimate value \( v \) for which \( 2^i \)th node is responsible

Mercury

- Need to establish links based on node-distance
  - Piece-wise linear approximation
  - Histogram
Histogram Maintenance

- Measure node-density locally
- Gossip about it!

Values

Node-density

Load Balancing

- Basic idea - leave-join
  - "light" nodes leave
  - Re-join near "heavy" nodes; split the range of the heavier node

Load Balancing

- Basic idea: leave-rejoin
- Steps
  - Find average, check if heavy or light
  - Light nodes perform a leave and rejoin

Multi-attribute Range Queries

- Send data to all rings
- Send query to only ring
"push" samples to $k_2$ nodes

Ashwin, 8/25/2004