Information Retrieval in Structured Networks

Haripriya Rajagopal

Talk Outline

1. Background
2. On the Feasibility of Peer-to-Peer Web Indexing and Search
3. The Case for a Hybrid P2P Search Infrastructure
4. ODISSEA-A Peer-to-Peer Architecture for Scalable Web Search and Information Retrieval
5. Discussion

Basic Concepts in Information Retrieval

1. Query = Set of Search Terms
2. Posting list of a word ‘w’ = list of documents containing the ‘w’, possibly with the frequency of the term occurrence in each document.
3. Inverted Index = Mapping of words to their posting lists
4. For multiset queries, intersection of posting lists of the words returned
5. Result set pruned using Ranking Factors:
   - Importance of documents
   - Frequency of search term
   - Proximity of search terms

Basic Concepts

1. Vector Space ranking model: Each term represents a dimension in document vector
2. Both terms and queries are n-dimensional vectors
3. Relevance of a document = similarity of document vector with query vector
4. Each term is weighted using TFxIDF technique

On the Feasibility of Peer-to-Peer Web Indexing and Search

Why Consider P2P Search Engines?
1. More Robust than centralized search
2. More resistant to censoring or manipulated rankings

P2P search engines can be utilized:
1. For search within the P2P network
2. For web search and data mining outside P2P network
**Fundamental Constraints**

- **Storage Constraints**
  - Consider a corpus size of 3 billion
  - 1000 words per document
  - Every document has unique docID
  - 20 byte hash of document content typically in a DHT
  - $3 \times 10^9 \times 1000$ docIDs
  - Inverted Index size: $6 \times 10^{13}$ bytes
  - 1000 queries per second

- **Communication Constraints**
  - Bandwidth consumed must fit within Internet's capacity
  - Assume backbone cross-section bandwidth = 100 GB
  - 1000 queries/sec
  - Per-query b/w = 10 Mt/s i.e. 1 MB (optimistic)
  - If data sent by query no more than size of document retrieved,
    - Per-query b/w=10KB(pessimistic)

**Basic Approaches**

- **Partition-by-document**
  - Documents divided among the hosts
  - Each peer maintains local inverted index for its set of docs
  - Query flooded to all peers
  - High latency
  - Suitable for proximity search
  - Bandwidth consumption comparatively low
  - Joining or leaving of peers does not affect others

- **Partition-by-keyword**
  - Responsibility for words divided amongst peers (DHT used to map words to peers)
  - Each peer stores posting lists for words it is responsible for
  - Multiword queries require posting lists to be sent over the network
  - Low latency
  - Not suitable for proximity search
  - Consumes high bandwidth
  - Dynamic changes more expensive

**Optimizations**

- **Test Data**
  - 1.7 million pages crawled from MIT

- **Caching**
  - Cache posting lists received
  - 38% reduction in communication cost for MIT query trace

- **Pre-computation**
  - Pre-compute and store intersection of posting lists
  - Not feasible for all term pairs.

  - Why?
    - Queries follow Zipf distribution
    - For MIT Data Set: precomputation of 3% of term pairs led to 50% reduction in term cost

**Compression**

- **Bloom Filters** represent a set compactly (false positives, no false negatives)
  - Node 'A' sends bloom filter of posting list to node 'B'
  - 'B' intersects the bloom filter with its posting list and sends back the resulting docIDs
  - 'A' filters out false positives
  - For MIT Data set – compression ratio of 13 achieved
  - For small result sets, do multiple rounds of Bloom intersections
  - Gives a further 30% improvement
**Compression**

- **Gap Compression**
  - Periodically remap docIDs from 160-bit hashes to dense numbers from 1 -> number of documents.
  - When is it effective?
  - For MIT data set: average compression ratio of 30
  - Advantages over bloom filters?
  - In conjunction with another technique, Adaptive Set Intersection, 30% improvement

**Clustering**

- Group similar documents together.
- Assign contiguous docID's to documents within same cluster.
- Clustering with gap compression with adaptive set intersection improves compression ratio to 75.

**Compromises**

- Incremental Intersection combined with Ranking
  - Users look at only top 'n' results
  - Posting lists sorted based on Ranking Functions
  - Top ranked documents are incrementally transferred
  - Effective when the intersection is big related to the posting lists
- Compromising P2P Structure
  - One copy of Inverted Index per ISP

**ODISSEA**

- P2P architecture for scalable web search and information retrieval
- Provides a distributed global indexing and query execution service
- Can be used for content residing inside or outside of a P2P network

**ODISSEA : Design Overview**

- Lower Tier:
  - Maintains a global index structure
  - Executes simple but general classes of search queries efficiently
- Upper layer interacts with P2P-based lower layer via two classes of clients
  - Update clients (e.g., crawler, web server)
  - Query clients (user implemented optimized query execution plan)

**Target Application**

- Search in P2P networks
- Search in large intranet environments
- Web Search: A powerful API supporting client-based search tools
- Search middleware
System Details

- Name is a URL, it is hashed to 80-bit ID for lookups
- To update the index, use underlying Pastry to transfer the posting
- Hierarchical search using groups. Large groups are split
- Each group is replicated for fault tolerance.
- Need for synchronization of replicated index structures during updates.

Ranking

- Ranking Function F:
  \[ F(D, Q) = \sum_{i=0}^{m-1} f(D, q_i) \]
  \[ F(D, Q) = \sum_{i=0}^{m-1} f(D, q_i) + g(D) \]
  \( m \) = no of query terms
  \( g(D) \) = query-independent value (eg Pagerank)

Odissea: Distributed Pruning Protocol

1. The node holding the shorter list (A), sends the first \( x \) postings to node B. Let \( r_{min} \) = smallest value of \( f(d, q_0) \) transmitted
2. B performs lookups into its own list to compute the total scores of the corresponding documents. Retain the top \( k \) documents. Let \( r_k \) be the smallest score among these
3. B transmits first \( x \) postings with \( f(d, q_1) > r_k - r_{min} \) together with total scores of \( k \) documents from (2)
4. A performs lookups into its own list for the postings received from B and determines the overall top \( k \) documents

Efficient Query Processing

DPP: Example for K=2 and x=3:

- A containing term \( q_1 \): \( d_1 \) (0.9), \( d_2 \) (0.8), \( d_3 \) (0.7), \( d_4 \) (0.69), \( d_5 \) (0.67)
- B containing term \( q_1 \): \( d_6 \) (0.6), \( d_7 \) (0.5), \( d_8 \) (0.4), \( d_9 \) (0.3), \( d_{10} \) (0.2), \( d_{11} \) (0.1)

- A to B: \( f_{min} = 0.7 \) (from \( d_2 \))
- B computes: \( f(D_1, Q) = \sum_{i=0}^{m-1} f(D, q_i) + g(D) \)

Top 2 documents:
1. \( d_1 \) (0.9), \( d_2 \) (0.8), \( d_7 \) (0.5), because \( f(d_1, q_2) > 0.4 \) together with \( d_1 \) (0.9), \( d_2 \) (0.8)
2. \( d_6 \) (0.6), \( d_7 \) (0.5), because \( f(d_6, q_1) > 0.4 \) together with \( d_1 \) (0.9), \( d_2 \) (0.8)

Efficient Query Processing

- Bloom Filters
- Use of hybrid partitioning where large inverted lists are split amongst nodes to get parallel computation
Efficient Query Processing

Evaluation of DPP

- 900 two-term queries selected form a set of over 1 million
- Testing corpus: 120 million web pages (1.8TB) that were crawled by their own crawler
- Value of x determined by experiments on TA
- Computation within nodes are not taken into account
- Communication costs and estimated times of DPP for the top-10 documents and standard cosine measure:

<table>
<thead>
<tr>
<th>Metric</th>
<th>Metric 20%</th>
<th>Metric 40%</th>
<th>Metric 80%</th>
<th>Metric 90%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Setup time</td>
<td>2.941</td>
<td>4.175</td>
<td>4.417</td>
<td>5.745</td>
</tr>
<tr>
<td># postings A B</td>
<td>1.468</td>
<td>2.981</td>
<td>4.413</td>
<td>5.745</td>
</tr>
<tr>
<td>Total bytes transferred</td>
<td>28.344</td>
<td>45.392</td>
<td>70.840</td>
<td>53.020</td>
</tr>
<tr>
<td>Setup time (2MBps)</td>
<td>1.055</td>
<td>1.567</td>
<td>1.974</td>
<td>1.988</td>
</tr>
<tr>
<td>Setup time (2MBps)</td>
<td>4.886</td>
<td>8.760</td>
<td>15.640</td>
<td>4.408</td>
</tr>
</tbody>
</table>

The Case for a Hybrid P2P Search Infrastructure

- P2P Search - What is the best design?
- Target Environment
  - Items stored in P2P network, queried by keywords
    - Replicas of items follow a long-tailed distribution
    - Popular items at head of distribution
    - Rare items at tail of distribution
    - Typical of P2P file-sharing environments

Design Choices

- Unstructured Networks
  - Queries are flooded for bounded number of hops
  - E.g., Gnutella and Kazaa
  - No guarantees on recall
- Structured Networks
  - Inverted Indexes on Distributed Hash tables (DHTs)
  - Inverted Lists indexed by keyword:
    - <"Britney"; Doc4, Doc5, Doc6, Doc9, Doc11, ...>
    - <"Quantum"; Doc1, Doc3>
  - Query execution
    - Queries query to all sites hosting keyword in query
    - Intersection of multiple Inverted Lists.
    - Guarantees perfect recall (absence of network failures)

Their Proposal: Hybrid Solution

Flooding-based Network

DHT (Index Rare Items)

- Flood-based network
- DHT (Index Rare Items)
- Very Few Results
- More Results
- Simpler alternative to either optimizing unstructured or structured networks.

Gnutella Network

- Crawl of Gnutella Network
  - Based on multiple crawlers from 30 vantage points on PlanetLab
  - ≥ 100,000 nodes, 20 million files
- Ultrapeer-based Topology
  - Queries flooded among ultrapeers
  - Leaf nodes shielded from query traffic

Gnutella Measurements

- "Quality" of Searches:
  - Recall (% of all relevant items retrieved)
  - Response Time (Latency) to 1st result
- Software utilized:
  - Modified the LimeWire Gnutella Client
    - Run as leaf or ultrapeer
    - Sniff Gnutella traffic
    - Inject queries and gather results
Gnutella Search Quality

- Reissue Gnutella queries
  - 30 LimeWire Ultrappeers on PlanetLab
  - 700 Gnutella queries at 3 different times
- Compute Query Recall
  - Each query issued simultaneously from 30 ultrapeers
  - Union of results from 30 ultrapeers -- an approximation of the ideal answer

Queries with Small Result Sets

Result Size CDF

Gnutella is good enough.

Query Latency

Summary of Measurements

- Searching on Gnutella
  - Highly effective for popular items
  - Less effective for rare items

DHT-based Search

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

- Disadvantages:
  - Constructing inverted lists is costly
  - So is intersecting inverted lists at query time
  - Back-of-envelope calculations shows infeasibility for large datasets (IPTPS '03). But ODISSEE claims otherwise.

- Feasible for querying rare items:
  - Queries over rare items ship 7x fewer inverted list entries compared to the average query
Hybrid Search

1. Hybrid = “Best of both worlds”
   - Flooding techniques for searching popular items
   - DHT for rare items

2. Identifying rare items
   - Query snooping
     - Items from previous queries that return few results.
   - Other techniques (ongoing work):
     - Term Frequency Statistics (single term, pairs)
     - Sample items on neighboring nodes

Results

1. Improved Response Time:
   - PIER returns first result in 10 seconds
     - 40 seconds in aggregate including 30 seconds timeout
   - Gnutella queries return first result in 65 seconds
   - 25 seconds (38%) reduction in latency

Results

1. Improved Recall: 18% reduction in the number of queries that eventually receive no results from Gnutella
   - Lots of room for improvement: 66% potential reduction based on “Union-of-30” query