Efficient Search - Overview

Improving Search In Peer-to-Peer Systems

Presented By Jon Hess
cs294-4 Fall 2003

Efficient Search - Overview

- Goals
  - Present alternative searching methods for systems with loose integrity constraints
    - Probabilistically optimize over loose constraints
    - Must not sacrifice user satisfaction
  - Measure hypothesized methods
  - Analyze the measurements

Efficient Search - Overview

- Current peer-to-peer systems
  - Offer efficient search, but
    - Restrict data placement
    - Restrict search breadth
  - Provide data consistency
  - Provide data availability

Efficient Search - Overview

- Current file-sharing systems do not require
  - Top notch data availability or consistency
  - Search by identifier
- So they do not need
  - Restrictive data, and pointer, placement
- So they may
  - Make probabilistic optimizations

Efficient Search - Overview

- Problem Overview
  - System should
    - Accept a query
    - Respond with a set of pointers to files
  - System is composed of an overlay of peers
    - Upon receiving a query, a peer
      - Processes the query
      - Responds with matches
      - Forwards query to neighbors

Efficient Search - Overview

- Cost Measurement
  - Processing Cost
    - Deciding which neighbors to route to
    - Checking local metadata for query matches.
  - Bandwidth Cost
    - Forwarding messages
    - Responding to queries
  - Calculate over a set of queries
  - Measure the aggregate network cost

Efficient Search - Overview

- Quality Measurement
  - Size of result set
    - Maximum of the fraction of number of results with relation to some goal Z
      - Min (Results / Z , 1.0)
    - Latency from query to Z^{th}, or final result
      - Min (time(Z) , time(final))

Efficient Search - Methods

- Current Technique
  - Gnutella - breadth first flood
    - Efficiencies
      - Maximizes number results
      - Minimizes latency
    - Inefficiencies
      - The same node may process the same query twice
      - Many more than Z results may be achieved

Efficient Search - Methods

- Current Technique
  - Freenet – serial depth first search
    - Efficiencies
      - Achieves exactly Z matches
    - Inefficiencies
      - Maximum Latency
      - No parallel searching

Efficient Search - Methods

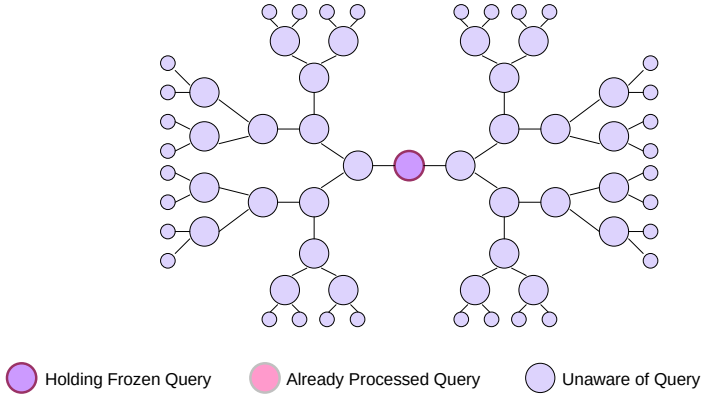
- Suggestions
  - Iterative deepening
    - Try not to locate more than Z matches
  - Directed breadth first flood
    - Don't traverse wasteful links
  - Nearest neighbor
    - Amortize processing cost across memory and neighbors
    - Does not need to make final hop[s] to search furthest neighbors

Efficient Search - Methods

- Iterative Deepening
  - Successively send deeper reaching queries until Z is satisfied
  - Policy: set of depths $\{d_0 \ldots d_n\}$ and a time t
  - Send query to depth d_i
    - Nodes at depth d_i freezes query after processing
    - If Z is satisfied stop
    - If not, send a re-query message to nodes at d_i instructing them to forward the query to d_{i+1}

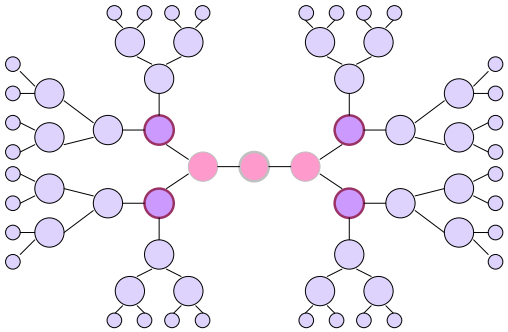
Efficient Search - Methods

- Iterative Deepening – {0, 2, 4, 5}, 3



Efficient Search - Methods

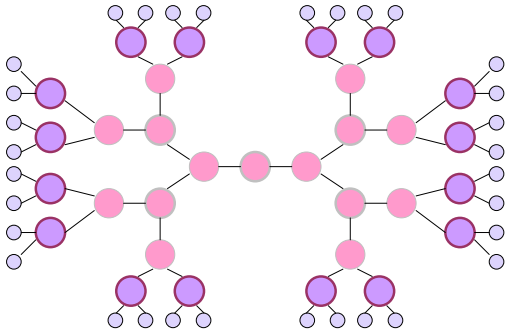
- Iterative Deepening – {0, 2, 4, 5}, 3



Holding Frozen Query Already Processed Query Unaware of Query

Efficient Search - Methods

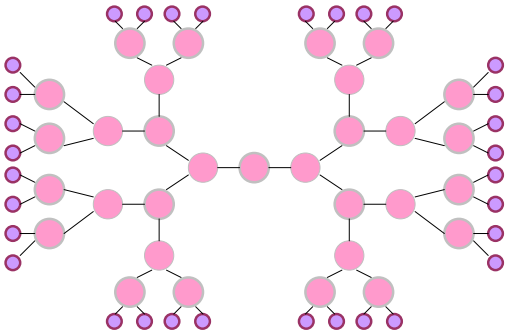
- Iterative Deepening – {0, 2, 4, 5}, 3



Holding Frozen Query Already Processed Query Unaware of Query

Efficient Search - Methods

- Iterative Deepening – {0, 2, 4, 5}, 3



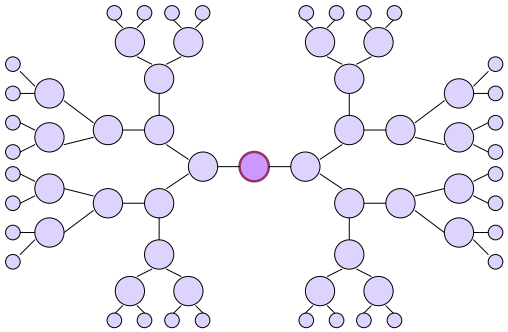
Holding Frozen Query Already Processed Query Unaware of Query

Efficient Search - Methods

- Directed Breadth First Search
  - Source Only sends queries to *good* neighbors
  - Good neighbors might have
    - Produced results in the past
    - Low latency
    - Lowest hop count for results
      - They have *good* neighbors
    - Highest traffic neighbors
      - They're stable
    - Shortest message queue
  - Routed as normal BFS after first hop

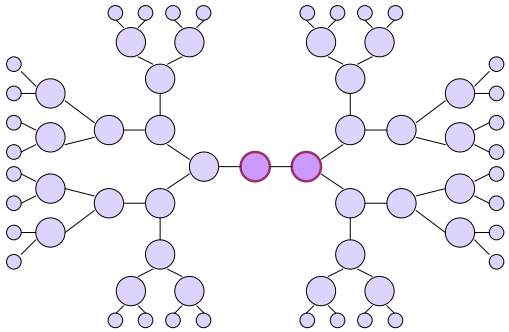
Efficient Search - Methods

- Directed Breadth First Search



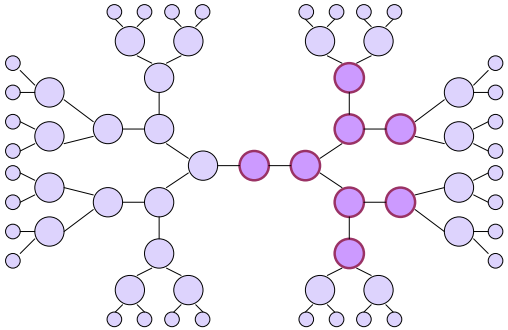
Efficient Search - Methods

- Directed Breadth First Search



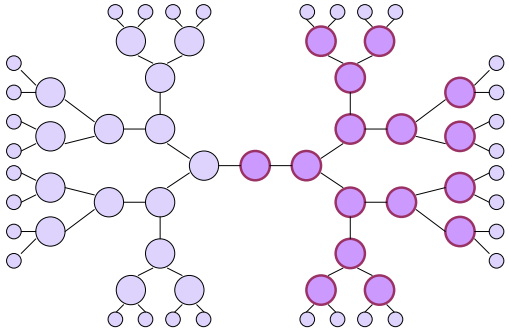
Efficient Search - Methods

- Directed Breadth First Search



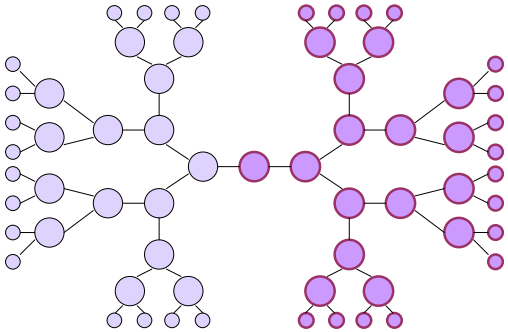
Efficient Search - Methods

- Directed Breadth First Search



Efficient Search - Methods

- Directed Breadth First Search

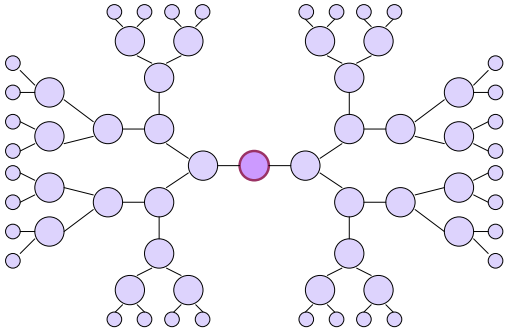


Efficient Search - Methods

- Neighbor Index
  - Policy: Set of query depths $\{d_0, \ldots, d_n\}$ and index depth r.
  - Each node holds an index of neighbor's metadata within n hops
  - Nodes propagate local metadata upon joining
  - Only nodes at depths d_i process queries
    - Other nodes simply forward until d_n

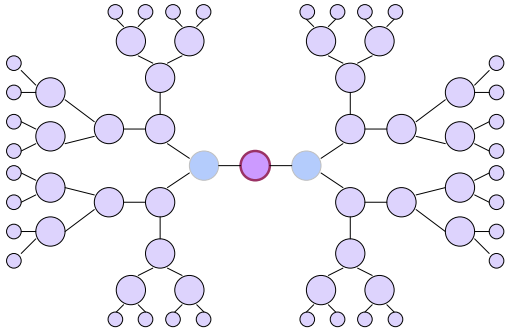
Efficient Search - Methods

- Neighbor Index – {0, 3, 5}, 1



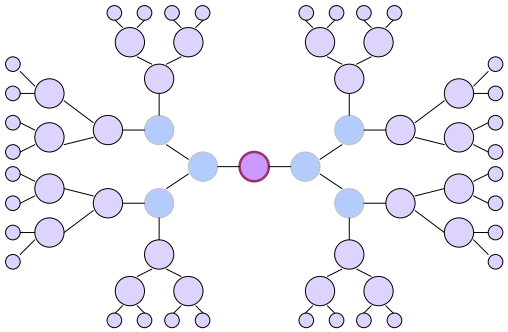
Efficient Search - Methods

- Neighbor Index – {0, 3, 5}, 1



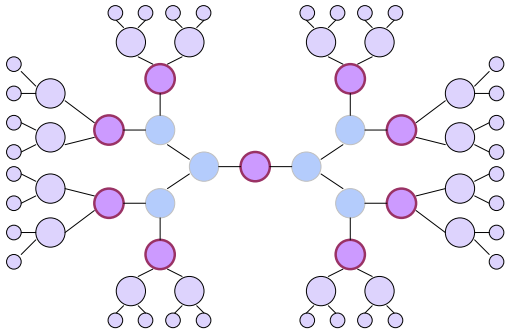
Efficient Search - Methods

- Neighbor Index – {0, 3, 5}, 1



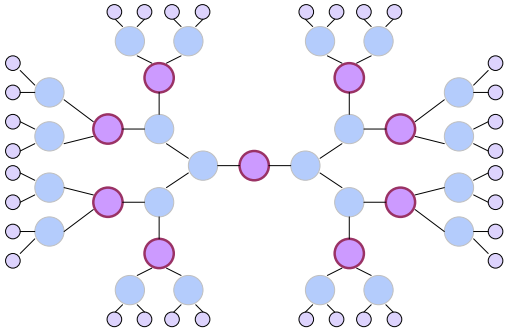
Efficient Search - Methods

- Neighbor Index – {0, 3, 5}, 1



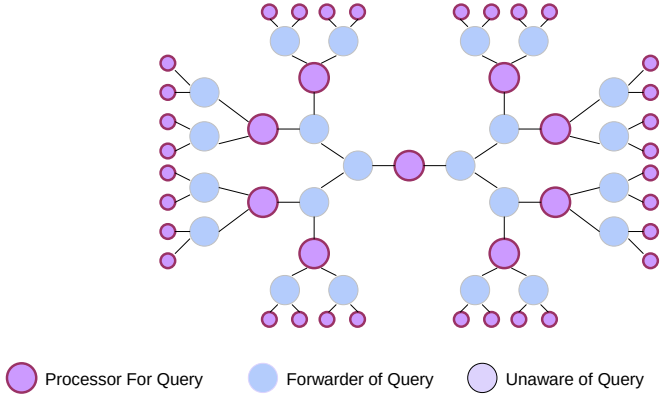
Efficient Search - Methods

- Neighbor Index – {0, 3, 5}, 1



Efficient Search - Methods

- Neighbor Index – {0, 3, 5}, 1



Efficient Search - Experiment

- Data Collection
  - Inject a custom client into the Gnutella network for 1 month
  - Collect statistics
    - Query relationships
    - Average files per user
    - Average length of messages
    - Many other statistics
  - Calculate the expected cost of the proposed methods from the harvested data

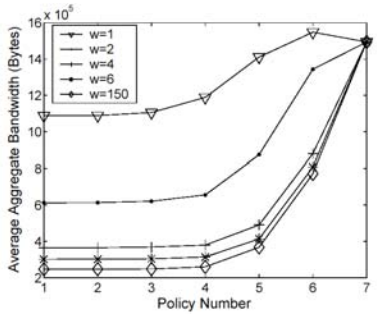
Efficient Search - Experiment

- Iterative Deepening & Neighbor Index
  - For each Query q and Depth d
    - Send Q with TTL d
  - See how many matches nodes at each depth possess
- Directed Breadth First Flood
  - Queries are sent to each neighbor once at a time, with a different query ID.

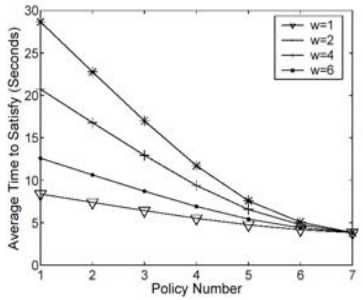
Efficient Search - Results

- Iterative deepening
  - Study the policies
    - P_1 = {1, 2, 3, 4, 5, 6, 7}
    - P_2 = {2, 3, 4, 5, 6, 7}
    - …
    - P_6 ={6, 7}
    - P_7 ={7}
    - For W = {1, 2, 4, 6, 150}

Efficient Search - Results



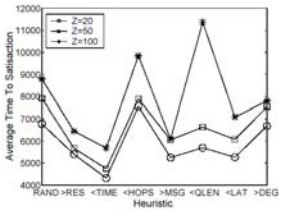
As freeze time decreases, or policy number increases, Z is overshot.



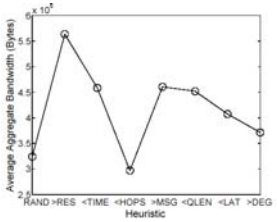
As freeze time decreases or policy number increases, we achieve Z faster.

If all nodes used this policy would average satisfaction time decrease?

Efficient Search - Results



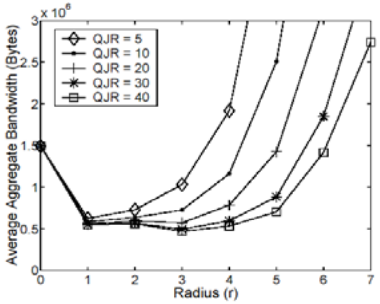
Shortest average time to satisfaction obviously wins. Surprisingly, the greatest number of results method does very well.



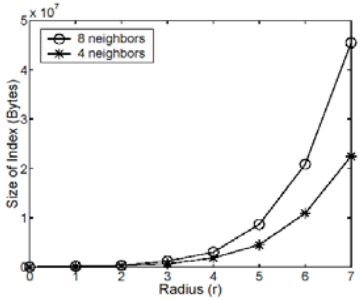
Greatest number of results, and *shortest time to satisfaction* do the best on the other benchmarks, but cost the most – obviously, results require bandwidth.

This cost is still less than BFS.

Efficient Search - Results



Query radius has exponential relationship to bandwidth cost.



Only a 5MB Index for peers 5 hops away.

If all nodes used this policy would average satisfaction time decrease?

Efficient Search - Questions

Questions?