Efficient search in peer to peer networks

Authors: Beverly Yang, Hector Garcia-Molina, in proceedings of the ICDCS'02 conference, 2002

Presented by: Venkata Gopal K Addada

Acknowledgements

Some of the followings slides are borrowed or adapted from talks at NetCINS lab, University of Patras, Greece.

Key Challenges

- Designing efficient techniques for search and retrieval of data in peer-peer systems
- Best search techniques for a system depends on the needs of the application.
- In structured P2P systems the retrieval of object is guaranteed, if it exists in the system.
- Current search techniques in "loose" P2P systems tend to be very inefficient, either generating too much load on the system, or providing for a very bad user experience.

Towards Efficient Searching

- Queries are processed by more nodes than desired.
  - Experiments show that most queries can be answered by querying fewer nodes
- Improve Search Techniques
  - Make queries more efficient
  - Generate as little load as possible
  - Provide good user experience
- Suggested Improvement
  - Processing queries through fewer nodes.

Techniques

- Iterative Deepening

  Iteratively send the query to more nodes until query is answered

- Directed BFS (Breadth First Search)

  Send the query to an intelligently selected set of nodes

- Local Indices

  Nodes maintain small indices over other nodes' stored data

Problem Framework 1/2

- P2P: Undirected graph
  Vertices: nodes in the n/w
  Edges: Open connections between neighbors.

- Messages will travel from node A to B following a path.

- Length of the path: Number of hops

- Source of query: Node submitting the query

Problem Framework 2/2

- When a node receives a query it should process the query locally and
  - respond to the query
  - forward the query or
  - drop the query

7/37

Metrics

- Cost
  - Average Aggregate Bandwidth
  - Average Aggregate Processing Cost
    for a representative set of queries, Q_{rep}
- Quality of Results
  - Number of Results
  - Satisfaction of the query
    the query is satisfied if Z or more results are returned
  - Time to satisfaction:
    how long the user must wait for the Z_{th} result to arrive

8/37

Proposed Search Techniques

- Gnutella
  Breadth-first traversal (BFS) over the network with depth limit D
- Freenet
  Depth-first traversal (DFS) over the network with depth limit D.

9/37

Discussion…

- Quality of results measured only by number of results then BFS is ideal
- If Satisfaction is metric of choice BFS wastes much bandwidth and processing power
- With DFS each node processes the query sequentially,searches can be terminated as soon as the query is satisfied, thereby minimizing cost.But poor response time due to the above
  (Worst case is exponenitial in D)

10/37

Broadcast Policy

- BFS and DFS falls on opposite extremes of bandwidth/processing cost and response time.
- Need to find some middle ground between the two extremes, while maintaining quality of the results.

11/37

Iterative Deepening

- When satisfaction is the metric of choice
- Multiple BFS are initiated with successively larger depths, until query is satisfied or the maximum depth limit *D* is reached
- System wide policy specifying at what depth the iterations are to occur
- A waiting period *W* (time between successive iterations in the policy) must be specified

12/37

Working of Iterative Deepening 1/2

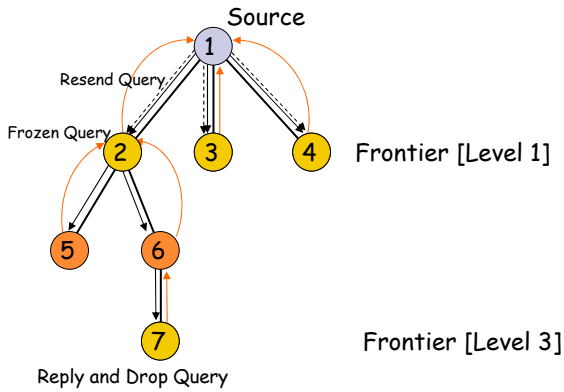
- Policy P{*a,b,c*}
- Source node *S* initiates a BFS of depth *a* by sending out a query message to all its neighbors
- Query becomes frozen at all nodes *a* hops away from *S* (Frontier nodes)
- *S* receives response from those nodes that have processed the query so far and waits for a time period W.
- If the query is not yet satisfied, *S* will start the next iteration, initiating BFS at depth *b by sending a Resend message.*

Working of Iterative Deepening 2/2

- A node that receives a resend message,simply unfreezes the query (stored temporarily) and forwards the query to its neighbors.
- This process continues in the similar fashion till depth *D* is reached. At depth *D*, the query is dropped
- Identifying Queries
  - Every query is assigned a system wide "*unique identifier*".
  - The resend message will contain the identifier of the query so as the frontier nodes will know which query to unfreeze.

Iterative Deepening

P={1,3}



Directed BFS

1/2

- When response time is the metric of choice.
- Send queries to a subset of nodes that will return many results quickly
- Statistics (History) about neighbors should be kept
- By sending the queries to a small subset of nodes:
  - the cost incurred will be reduced significantly
  - The quality of results is not decreased significantly

Directed BFS

2/2

- Criteria for selecting the best neighbor
  - Returned most results
  - Shortest satisfaction time
  - Min hops for results
  - Sent us most messages (all types)
  - Shortest Message queue
  - Shortest latency
  - Highest degree

17/37

Local Indices

1/5

- A node maintains an index over the data of each node within *r* hops of itself
- When a node receives a Query message, it can then process the query on behalf of every node within *r* hops of itself
- Collections of many nodes can be searched by processing the query at few nodes, while keeping the cost low

18/37

Local Indices

2/5

- Radius *r* is a system-wide parameter
- *r* should be small.
- The index will be small - typically of the order of 50 KB- independent of the size of the network

19/37

Local Indices

3/5

- Policy specifies at which depth query will be processed *ex:* P = { a,b,c }
- All nodes at depths not listed in the policy will simply forward the query to the next depth
- Last value in policy P (*c* in above example) can have maximum value of (D-r). (Why?)

20/37

Local Indices

4/5

- When a new node joins: Sends a join message with TTL=*r* and all the nodes within *r* hops update their indices.
- Join message contains the metadata about the joining node
- When a node receives this join message it, in turn, send join message containing its meta data directly to the new node.New node updates its indices

Local Indices

5/5

- Node dies: Other nodes update their indices based on the *timeouts*
- Updating the node: When a node updates its collection, his node will send out a small update message with TTL= *r, containing the metadata* of the affected item.All nodes receiving this message subsequently update their index.

Experimental Setup

- Existing GNUTELLA clients are used
- Representative set of queries Q_{rep} used to analyze the results
  (500 from 500,00 observed queries)
- GNUTELLA 'PING' messages used to calculate number of hops to a node
- Experiments only performed for 'Iterative deepening' and 'Directed BFS'

Metrics

- Average Aggregate Bandwidth: The average, over a set of representative queries Q_{rep}, of the aggregate bandwidth consumed (in bytes) over every edge in the network on behalf of each query.
- Average Aggregate Processing Cost: The average, over a set of representative queries Q_{rep}, of the aggregate processing power consumed at every node in the network on behalf of each query.

Results for Iterative Deepening 1/4

- Policies used for analysis:

  $P = \{ P_d = \{ d, d+1, \ldots, D\}$, for $d = 1,2\ldots.,D\}$
- $P_1 = \{1, 2, 3, \ldots., D\}$

  $P_2 = \{2, 3, \ldots., D\}$

  $P_3 = \{3, 4, \ldots., D\}$

  .

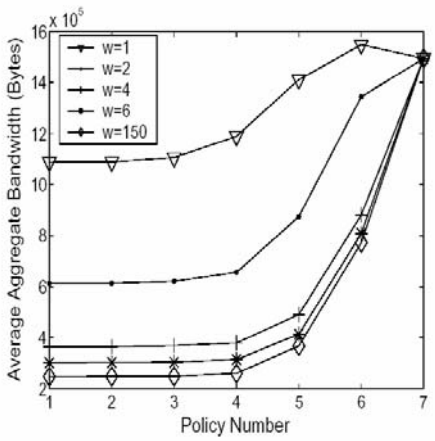
  .

  $P_{D-1} = \{D-1, D\}$

  $P_D = \{D\}$

25/37

Results for Iterative Deepening 2/4



Bandwidth consumption for various iterative deepening policies

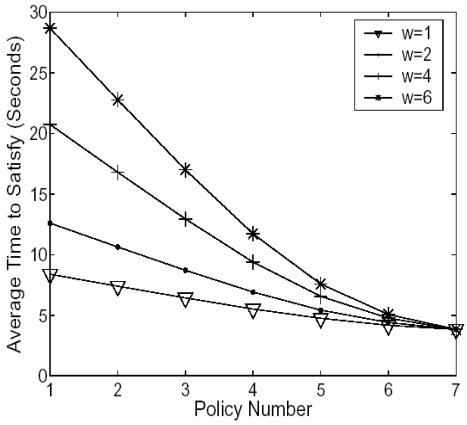
- Bandwidth cost increases as d increases

  Sending the query to more nodes than necessary will generate extra bandwidth consumption. (Remember Z=50 across all experiments)
- Bandwidth cost increases as W decreases

  if W is small there is higher likelihood that the source will determine that the query was not satisfied (Prematurely)
- Authors recommended P_5 and W=6

26/37

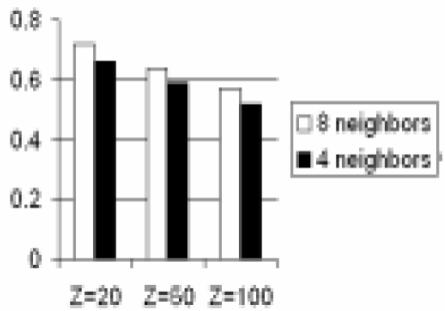
Results for Iterative Deepening 3/4



Time to satisfaction for various iterative deepening policies

- Time to satisfaction increases as d decreases
  as d decreases the number of iterations needed to satisfy a query will increase.
- Time to satisfaction increases as W increases
  as W decreases the time spent at each iteration decreases and thus the time to satisfaction decreases too.

Results for Iterative Deepening 4/4



Probability of satisfaction for different Z

- Satisfaction with 4 neighbors is not much lower than satisfaction with 8 neighbors
- Authors suggest NOT to have large number of neighbors

Results for Directed BFS

1/3

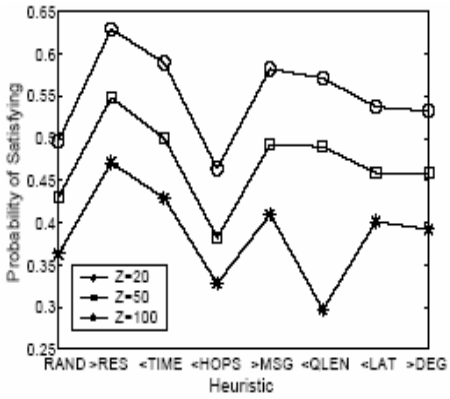
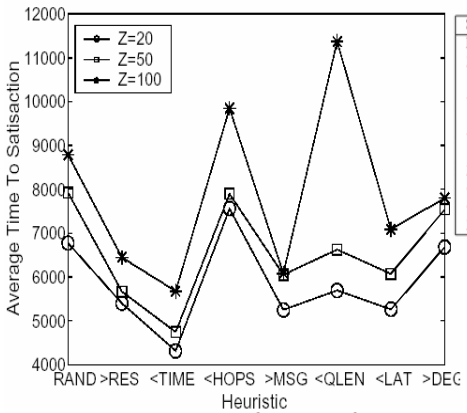


Figure 3. Probability of satisfaction for Directed BFS policies

- >RES is the best one followed by <TIME
- <HOPS is worse than RAND
- Authors could not explain why performance of <QLEN drops when Z = 100

Results for Directed BFS

2/



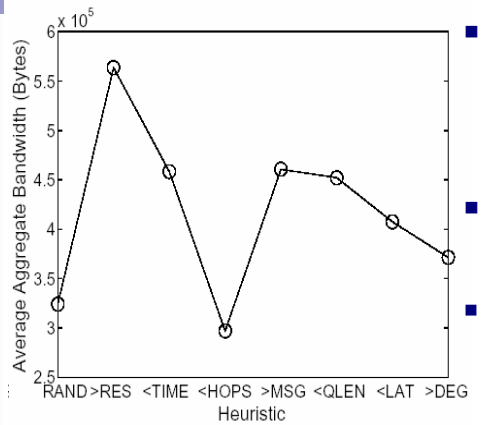
Time to satisfaction for various Directed BFS policies

Symbol	Heuristic: Select neighbor that...
RAND	(Random)
>RES	Returned the greatest number of results in the past 10 queries
<TIME	Had the shortest average time to satisfaction in the past 10 queries
<HOPS	Had the smallest average number of hops taken by results in the past 10 queries
>MSG	Sent our client the greatest number of messages (all types)
<QLEN	Had the shortest message queue
<LAT	Had the shortest latency
>DEG	Had the highest degree (number of neighbors)

- <TIME is the best one followed by >RES
- >DEG does not perform as expected

Results for Directed BFS

3/



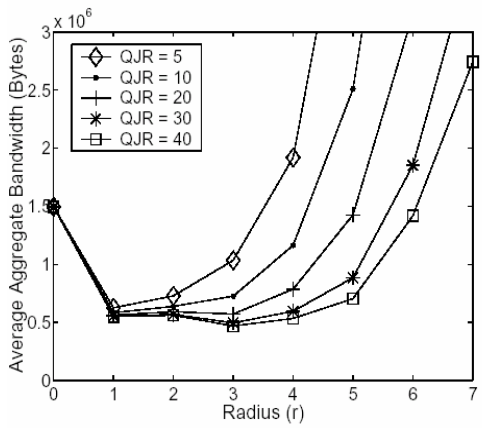
Bandwidth consumption for Directed BFS

- There is a correlation between cost and quality of results

  More quality results implies more aggregated bandwidth
- Bandwidth consumption independent of Z
- Iterative Deepening vs Directed BFS
  - Directed BFS performs better when looking at time to satisfaction
  - Iterative deepening can achieve lower cost

Results for Local Indices

1/3

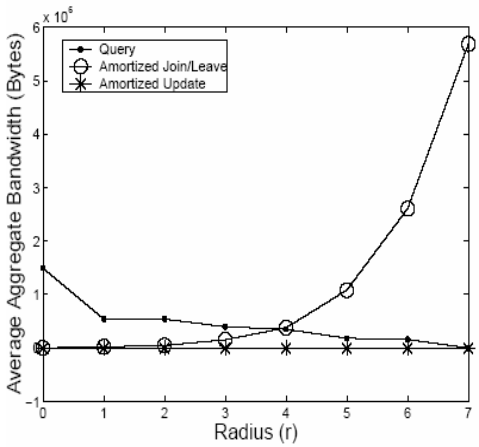


Policies
$P_0 = \{1, 2, 3, 4, 5, 6, 7\}$
$P_1 = \{0, 3, 6\}$
$P_2 = \{0, 5\}$
$P_3 = \{4\}$
$P_4 = \{3\}$
$P_5 = \{2\}$
$P_6 = \{1\}$
$P_7 = \{0\}$

Bandwidth consumption for Local Indices

- As QueryJoinRatio increases the cost decreases.
- The cost of node joins/leaves dominates the query cost for large values of r
- For a normal system with QJR=10 the best choice is r=1.

Results for Local Indices 2/3



Comparison of Bandwidth Consumed by Queries and Join/Leaves

- Query/Join ratio = 20
- Cost of joins/leaves grows exponentially

  When 'r' is large this cost dominates over the cost of queries
- Amortized cost of updates is always relatively small fraction of total cost

33/37

Results for Local Indices 3/3

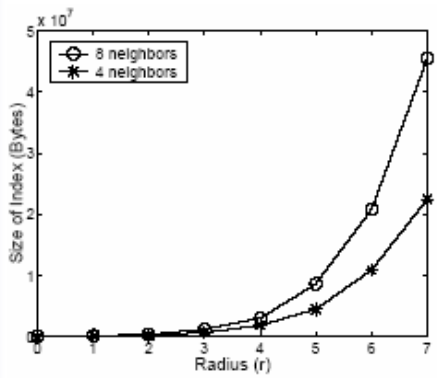


Figure 9. Size of the Index for different Radii(r)

- Cost:
- Even though the size of the index grows exponentially, it still practical for all r in range. For example, at r = 7 with 4 neighbors, the size of the index would be roughly 21MB. For r =1, the size of the index would be roughly 71KB.

34/37

Conclusions

1/2

- Compared to BFS the discussed techniques greatly reduce the aggregate cost of processing query over the entire system, while maintaining the quality of results
- Schemes are simple and practical to implement on the existing systems
- Bootstrapping new node in directed BFS scheme is not well-defined (No statistical data/History)

Conclusions

2/2

Technique	Time to Satisfy	Probability of Satisfaction	Number of Results	Aggregate Bandwidth	Aggregate Processing
BFS	100%	100%	100%	100%	100%
Iterative Deepening ($d = 5, W = 6$)	190%	100%	19%	28%	47%
Directed BFS (>RES)	140%	86%	37%	38%	28%
Local Indices ($r = 1$)	$\approx$ 100%	100%	100%	39%	51%

Relative performance technique using BFS as baseline

Questions

?

37/37