

ODISSEA: a Peer-to-Peer Architecture for Scalable Web Search and IR

Torsten Suel

with C. Mathur, J. Wu, J. Zhang,
A. Delis, M. Kharrazi, X. Long, K. Shanmugasunderam

CIS Department
Polytechnic University
Brooklyn, NY 11201

<http://cis.poly.edu/westlab/odissea/>
(google: "odissea peer")

Talk Outline:

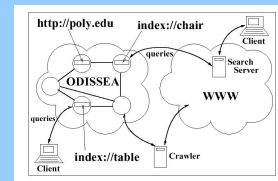
- ODISSEA: architecture, motivation, ideology
 - system design
 - discussion of design choices
 - our vision: open distributed web search architecture
- Distributed query processing
 - query execution in large search engines
 - efficient distributed top-k queries
 - experimental results
- Open problems and future work

Introduction:

- huge amount of work on web search
- huge amount of activity in P2P
- so, how about P2P (full text) search?
 - to query content in P2P networks
 - to query content located outside P2P network
- current engines based on scalable PC clusters
- so are many other "giant scale services"
- we know how to do file sharing in P2P
- how about search engines and large-scale IR?

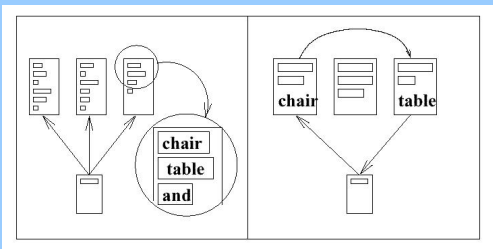
ODISSEA:

"Open DIStributed Search Engine Architecture"



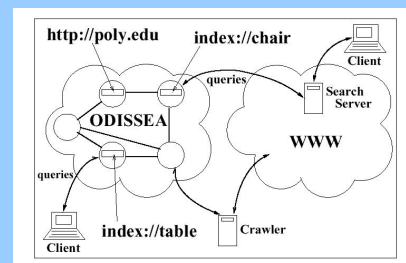
- global indexing and query execution service
 - scalable to size of the web
 - scalable to large query load
 - highly robust
 - open

Global index organization:



- avoids broadcasting query to all nodes
- faces other problems: updates, long inverted lists
- our main technical focus: efficient top-k queries

Two-tier architecture:



- scalable lower tier for indexing and query execution
- crawling outside system
- open interface supporting client-based tools

Applications:

- search of content located in P2P network
- distributed search in large organizations
- as a large-scale web search engine
- as global search middleware on top of system of local index structures

Vision: open web search infrastructure

- beyond current web search:
 - smart desktop-based search tools
 - browsing assistants, navigational toolbars
 - access lower-level search infrastructure
- can we have a common infrastructure?
 - open
 - scalable
 - agnostic
- example: Google API (not really)
- discussion: “entry barrier to search”
- tradeoff/challenge: performance vs. flexibility

Discussion: P2P and massive data

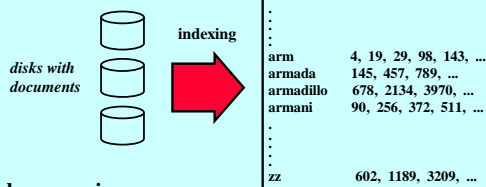
- P2P system spectrum:
 - unstructured (Gnutella etc) vs. structured (DHT)
 - rapidly evolving vs. fairly static
- massive data apps = fairly static system?
 - limit to how fast we can move data around
 - exception: file sharing (download, then share)
- we are at the more stable end of spectrum
- failures vs. unavailability
- replication and synchronization challenges

Implementation:

- based on Pastry DHT
- index and objects stored in Berkeley DB
- fine-grained postings traffic via P2P links
- replication for fault-tolerance
- replication based on “object groups”
- nodes may be temporarily unavailable
- synchronization of nodes upon reentry

Query processing in search engines

- inverted index
 - a data structure for supporting text queries
 - like index in a book



Boolean queries:

(zebra AND armadillo) OR armani

➔ unions/intersections of lists

Ranking in search engines:

- scoring function: assigns score to each document with respect to a given query
- top-k queries: return k documents with highest score
- example cosine measure

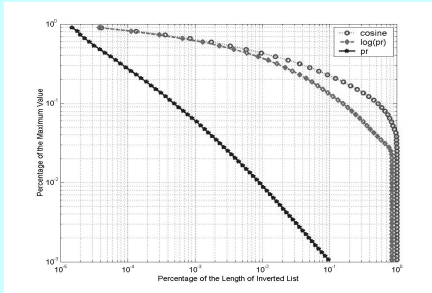
$$F(d, t_0, \dots, t_{m-1}) = \sum_{i=0}^{m-1} \frac{w(q, t_i) \cdot w(d, t_i)}{\sqrt{|d|}},$$
$$w(q, t) = \ln(1 + N/f_t), \text{ and}$$
$$w(d, t) = 1 + \ln f_{d,t},$$

- term-based vs. link-based ranking

$$F(x) = \sum_{i=0}^{m-1} f(d, q_i) \quad \text{and} \quad F(x) = g(d) + \sum_{i=0}^{m-1} f(d, q_i).$$

- many other important factors (links, user feedback, \$, markup)

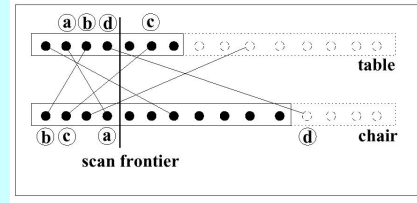
Using Pagerank in ranking:



- how to combine/add pagerank score and cosine? (addition)
- use PR or log(PR) ?
- normalize using mean of top-100 in list (Richardson/Domingo)

Efficient algorithms for top-k queries:

- recent work by Fagin and others
- FA (Fagin's algorithm), TA (Threshold algorithm), others
- term-based ranking: presort each list by contribution to cosine



- Pagerank: (pre)sort by combination of cosine and Pagerank?

Some results:

	top1	top4	top10	top100
Lists	879,010	879,010	879,010	879,010
Intersect	786	1,171	1,705	8,033
FA (cosine only)	4,638	8,012	12,445	39,306
TA (cosine only)	1,114	2,199	3,441	13,014
CA (cosine only)	371	931	1,729	9,429
FA (cosine + pagerank)	3,038	5,083	8,453	34,029
TA (cosine + pagerank)	677	1,346	2,255	11,333
CA (cosine + pagerank)	245	578	1,075	8,075
FA (cosine + log(PR))	944	1,410	2,137	9,709
TA (cosine + log(PR))	228	373	625	3,721
CA (cosine + log(PR))	97	166	288	2,102

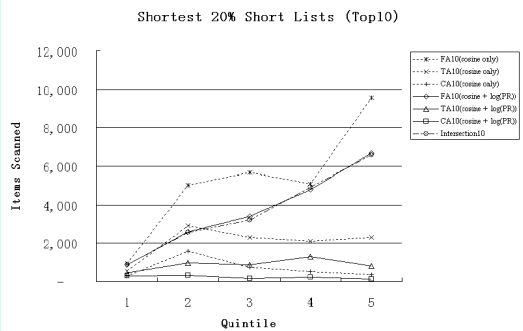
- centralized setting
- 120 million crawled pages
- Excite query trace
- CA = "clairvoyant algorithm"

More details:

Lists	shortest 20%		shorter 20%		middle 20%		longer 20%		longest 20%	
	top10	top100	top10	top100	top10	top100	top10	top100	top10	top100
FA (cosine only)	5,298	7,452	16,115	34,786	17,361	59,105	13,541	52,214	10,017	43,058
TA (cosine only)	2,057	4,978	4,083	15,304	2,904	12,677	4,417	16,942	3,754	15,244
CA (cosine only)	705	3,031	2,251	9,609	1,285	9,332	2,495	13,126	1,916	12,131
FA (cos + log(PR))	3,691	7,582	3,990	21,730	1,672	11,109	865	5,623	498	2,615
TA (cos + log(PR))	889	3,922	922	6,762	610	3,587	407	2,834	304	1,534
CA (cos + log(PR))	227	1,696	403	3,283	352	2,309	261	2,032	197	1,200

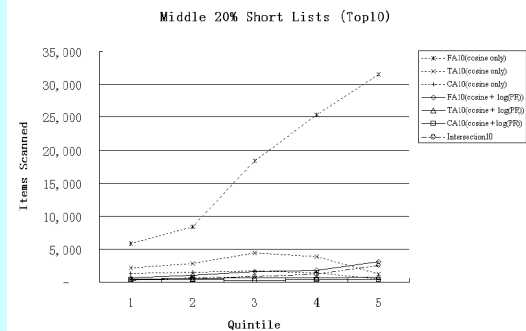
- most savings for long lists
- in fact, cos + log(PR) schemes get better and better

Shortest shorter lists:



- some methods increase with length of other list
- intersection pretty bad

Medium shorter lists:

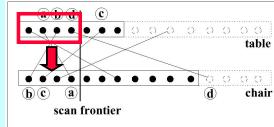


- only FA with cosine increases with length of longer list
- others much better and closer to each other

Distributed implementation:

- (1) The node holding the shorter list, called node *A*, sends the first *x* postings of its inverted list to node *B*. (Let's assume for the moment that *A* somehow knows the best value of *x*.) Also, let r_{min} be the smallest (last) value $f(d, q_i)$ transmitted.
- (2) Node *B* receives the postings from *A*, and performs a lookup into its own list in order to compute the total scores of the corresponding documents. Retain the *k* documents with the highest score among these. Let r_k be the smallest score among these documents.
- (3) Node *B* now transmits to *A* all postings among its first *x* postings with $f(d, q_i) > r_k - r_{min}$, together with the total scores of the *k* documents from step (2).
- (4) Node *A* now performs lookups into its own list for the postings received from *B*, and determines the overall top *k* documents.

- one round-trip
- need to decide right length of prefix to send
- can be extended to more than two keywords



Results of distributed implementation:

- top-10 queries
- cosine (top) and cos + log(PR) (bottom)
- 8 bytes per posting
- TCP performance model for congestion window
- prefix length determined by threshold algorithm (TA)

	shortest 20%	shorter 20%	middle 20%	longer 20%	longest 20%
Lists	10,401	63,853	222,948	666,717	3,371,176
Number of postings sent from A to B	2,057	4,083	2,904	4,417	3,745
Number of postings sent from B to A	1,486	4,084	2,891	4,413	3,745
Total bytes transferred	28,344	65,336	46,360	70,640	59,920
Total communication time (400 Kbps)	1,052	1,477	1,216	1,550	1,405
Total communication time (2 Mbps)	833	1,368	1,107	1,441	1,295
Lists	10,401	63,853	222,948	666,717	3,371,176
Number of postings sent from A to B	889	922	610	407	304
Number of postings sent from B to A	792	923	612	407	307
Total bytes transferred	13,448	14,760	9,776	6,512	4,888
Total communication time (400 Kbps)	720	757	648	463	426
Total communication time (2 Mbps)	612	648	538	353	317

Related Work:

- P2P search: JXTA, pSearch, FASD, planetP, others
- with global index structure:
 - Gnawali (Chord)
 - Reynolds/Vahdat: Bloom filters
 - Li et al: feasibility of P2P search engines, Bloom filters and other techniques (IPTPS 2003)
- Pruning techniques for top-k queries
 - DB Community: Fagin et al. 1996 - now
 - IR Community: since 1980s (Buckley/Lewit SIGIR 85)
 - Persin/Zobel/Sacks-Davis 1996, Anh/Kretser/Moffat 2001
 - differences: random lookups, # of terms, AND vs. OR

Current Status and Future Work:

- system still being built (very basic version done)
- working on query optimization
 - integrating Bloom filters and other heuristics
 - optimizing query plans for 2 and more keywords
 - use of statistics
- loose ends in evaluation
 - results for three and more terms
 - integrating other measures (e.g., term distance)
- replication, synchronization

more info: <http://cis.poly.edu/westlab/odissea/>
(google: "odissea peer")