(Query) History Teaches Everything, Including the Future

Fabrizio Silvestri

High Performance Computing Laboratory
Istituto di Scienza e Tecnologie dell'Informazione (ISTI)
Consiglio Nazionale delle Ricerche (CNR)
Pisa, Italy
f.silvestri@isti.cnr.it

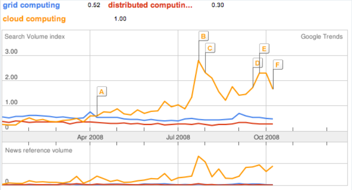
October 29th, 2008

Outline

What is History in our Case?

- Past Queries
- Query Sessions
- Clicktrough Data

From Google Trends



Our Main Data Source: Query Logs

- Store history about users search activity
- It is an extremely sensitive data
- Some publicly available logs are online
  - Excite (1997, 1999)
  - Altavista (2001)
  - AOL!!!
  - Microsoft Live! Log (see WSCD 2009)

What does a Query Look Like?

Some Examples

- “why is my husband so talkative with my female friends”

What does a Query Look Like?

Some Examples

- "why is my husband so talkative with my female friends"
- "can you hear me out there i can hear you i got you i can hear you over i really feel strange i wanna wish for something new this is the scariest thing ive ever done in my life who do we think we are angels and airwaves im gonna count down till 10 52 i can"

What does a Query Look Like?

Some Examples

- “why is my husband so talkative with my female friends”
- “can you hear me out there i can hear you i got you i can hear you over i really feel strange i wanna wish for something new this is the scariest thing ive ever done in my life who do we think we are angels and airwaves im gonna count down till 10 52 i can”
- “where is my computer”

What Topics are Represented

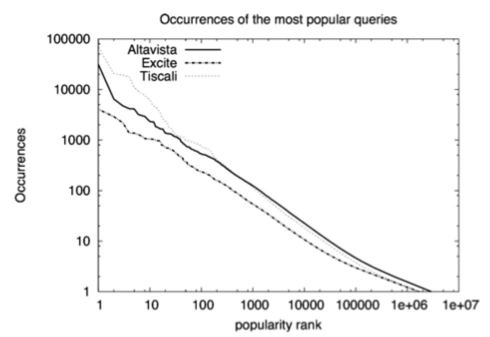
Distribution of Queries

Topic	**Percentage**
Entertainment	13%
Shopping	13%
Porn	10%
Research & learn	9%
Computing	9%
Health	5%
Home	5%
Travel	5%
Games	5%
Personal & Finance	3%
Sports	3%
US Sites	3%
Holidays	1%
Other	16%

From [Beitzel *et al.*, 2007]

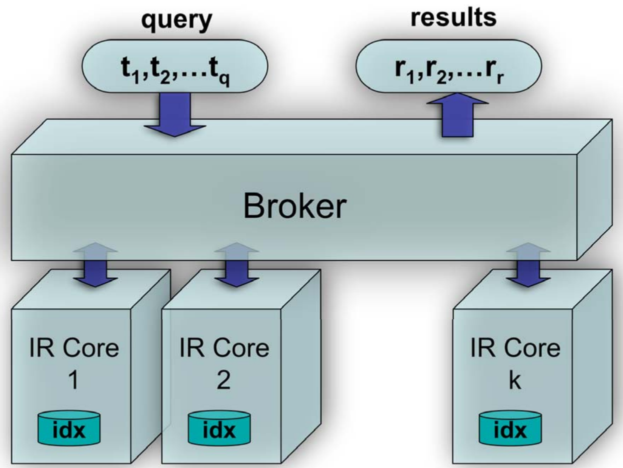
Power-laws in Query Logs

Query Distribution from a Yahoo! Search Engine Log

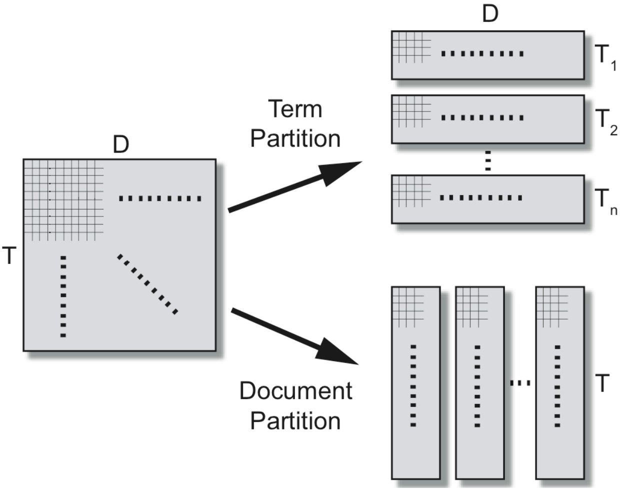


From [Fagni *et al.*, 2006]

The Architecture of a Distributed Search Engine

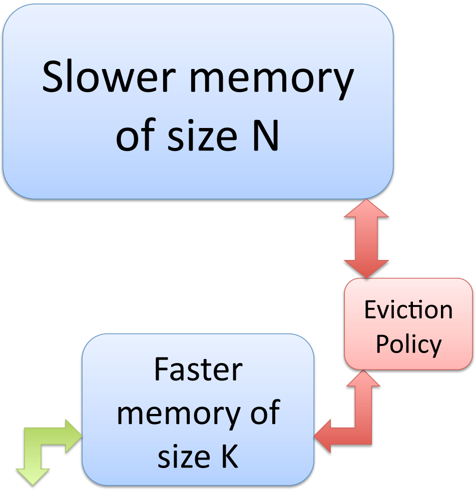


Data Partitioning



Outline

What is Caching?



Caching Goals

- Increase *Hit Ratio*
- Increase *Throughput*

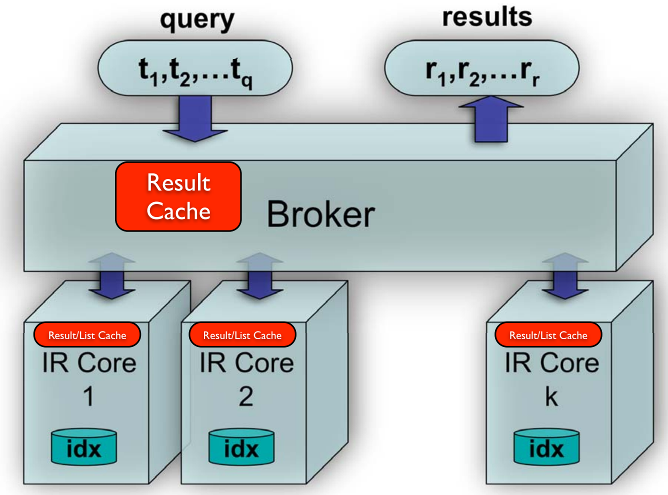
Hit Ratio

The ratio between the number of requests satisfied by the cache and the number of requests issued.

Throughput

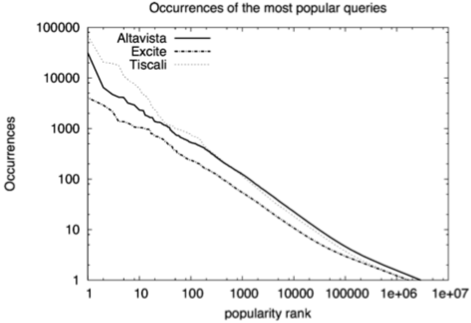
The number of requests answered in a time unit, e.g. *query-per-second*.

Cache Placement in Web Search Engines



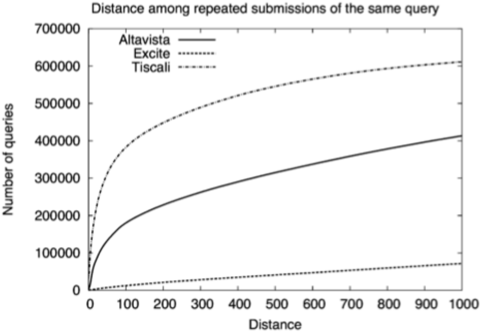
Is it worthwhile?

Consider again the power-law...



Is it worthwhile?

and now the distance between resubmission of the same query



Eviction Policies

Traditional

- LRU
- LFU
- ...
- see Markatos' work in [Markatos, 2000]

Related to Search

- Lempel and Moran PDC [Lempel and Moran, 2003]
- Fagni *et al.* SDC [Fagni *et al.*, 2006]
- Baeza-Yates *et al.* AC [Baeza-Yates *et al.*, 2007b]

History Based Caching

The Idea

To exploit the power-law to boost up past frequent queries (i.e. the head of the curve)

- Static based caching: was shown to be perform poorly by Markatos in [Markatos, 2000]

History Based Caching

The Idea

To exploit the power-law to boost up past frequent queries (i.e. the head of the curve)

- Static based caching: was shown to be perform poorly by Markatos in [Markatos, 2000]
- Probability Driven Caching scored queries on the basis of their likelihood to be seen in the future [Lempel and Moran, 2003]

History Based Caching

The Idea

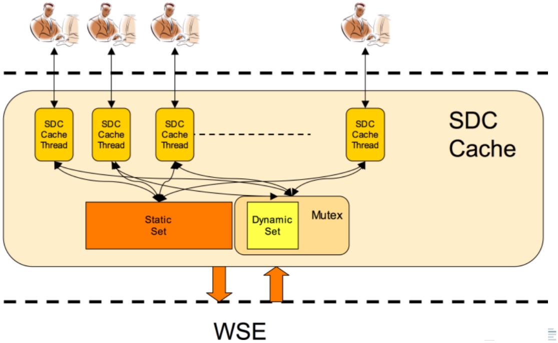
To exploit the power-law to boost up past frequent queries (i.e. the head of the curve)

- Static based caching: was shown to be perform poorly by Markatos in [Markatos, 2000]
- Probability Driven Caching scored queries on the basis of their likelihood to be seen in the future [Lempel and Moran, 2003]
- Static-Dynamic Caching (SDC): mixed up benefit from both static and classical (i.e. dynamic) caching (e.g. LRU) [Fagni *et al.*, 2006]

Static-Dynamic Caching

The Idea

Partition the cache into two parts. A *statically* filled part with the most frequently submitted in the past queries. A *dynamically* managed part using traditional policies (e.g. LRU)

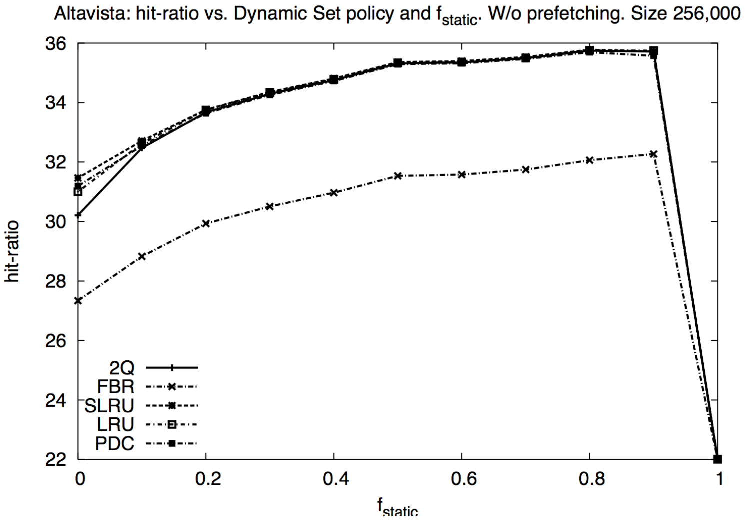


Test Collection

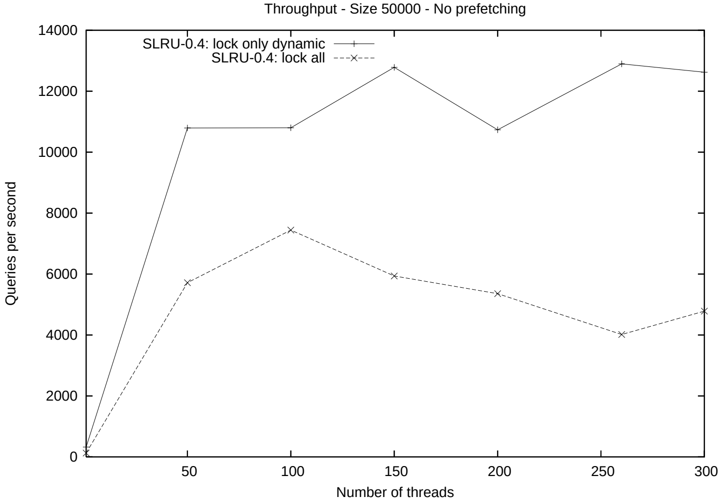
Main characteristics of the query logs used

Query log	queries	distinct queries	date
Excite	2,475,684	1,598,908	September 16^{th} 1997
Tiscali	3,278,211	1,538,934	April 2002
Alta Vista	7,175,648	2,657,410	Summer of 2001

SDC Hit-ratio



The Real Gain



Posting List Caching

The Idea

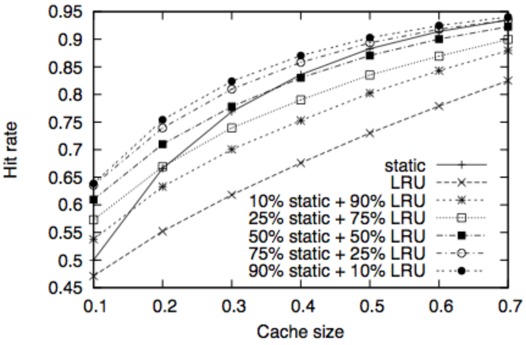
Instead of caching the result page for a complete query cache postings of its composing terms. E.g. For the query LA-Web Conference, LA, Web and Conference postings will be cached separately

- Traditional policies applied to lists. Correia Saraiva *et al.* [Correia Saraiva *et al.*, 2001]
- More refined policies based on a knapsack-like approach. Baeza-Yates *et al.* [Baeza-Yates *et al.*, 2007a]

Knapsack-like Caching

The Idea

Postings are variable-size. Keep in cache *frequently asked* but *not so big* posting lists.



Issues not covered by this talk

- Prefetching: anticipating users' clicks on the "*Next*" button [Fagni *et al.*, 2006]

Issues not covered by this talk

- Prefetching: anticipating users' clicks on the "*Next*" button [Fagni *et al.*, 2006]
- Sizing the posting and result cache [Baeza-Yates *et al.*, 2007a]

Issues not covered by this talk

- Prefetching: anticipating users' clicks on the "*Next*" button [Fagni *et al.*, 2006]
- Sizing the posting and result cache [Baeza-Yates *et al.*, 2007a]
- Theoretical analysis of trade-offs in query log caching [Baeza-Yates *et al.*, 2007a]

Lesson Learned

Using history allows us to...

- Detect "evergreen" queries (i.e. frequently repeating)

Lesson Learned

Using history allows us to...

- Detect "evergreen" queries (i.e. frequently repeating)
- Use these frequent queries to devise effective caching strategies (i.e. SDC)

Lesson Learned

Using history allows us to...

- Detect "evergreen" queries (i.e. frequently repeating)
- Use these frequent queries to devise effective caching strategies (i.e. SDC)
- Understand that the past is not always as the future (i.e. the Dynamic Set in SDC)

Lesson Learned

Using history allows us to...

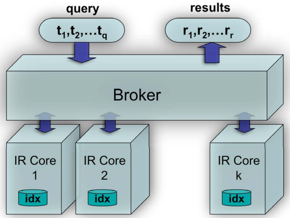
- Detect "evergreen" queries (i.e. frequently repeating)
- Use these frequent queries to devise effective caching strategies (i.e. SDC)
- Understand that the past is not always as the future (i.e. the Dynamic Set in SDC)
- Not shown... design *adaptive prefetching*, see [Fagni *et al.*, 2006]

Outline

The Architecture of a Distributed Search Engine... Again!

Usual Set Up

- Documents are partitioned assigning randomly a doc to each partition
- Queries are broadcasted to every IR Core



Can be something different done?

- Non randomly partition documents and non broadcast queries...

Can be something different done?

- Non randomly partition documents and non broadcast queries...
  - **Document Prioritization** [Puppin *et al.*, 2009]

Can be something different done?

- Non randomly partition documents and non broadcast queries...
  - **Document Prioritization** [Puppin *et al.*, 2009]
- Term Partitioning...

Can be something different done?

- Non randomly partition documents and non broadcast queries...
  - **Document Prioritization** [Puppin *et al.*, 2009]
- Term Partitioning...
  - **Smart Term Partitioning** [Lucchese *et al.*, 2007]

Document Prioritization

The Idea

Don't split documents randomly but cluster them in partitions according to how they appear together in search result pages.

The Overall Picture

- Collect associations query $\leftrightarrow$ retrieved documents

Document Prioritization

The Idea

Don't split documents randomly but cluster them in partitions according to how they appear together in search result pages.

The Overall Picture

- Collect associations query $\leftrightarrow$ retrieved documents
- Compute document similarities according to queries answered in common

Document Prioritization

The Idea

Don't split documents randomly but cluster them in partitions according to how they appear together in search result pages.

The Overall Picture

- Collect associations query $\leftrightarrow$ retrieved documents
- Compute document similarities according to queries answered in common
- Divide collection according to document similarities

Computing Document Similarity

- Can be done using a "*traditional*" clustering algorithm
- We applied co-clustering to the Query-Vector matrix M

Definition

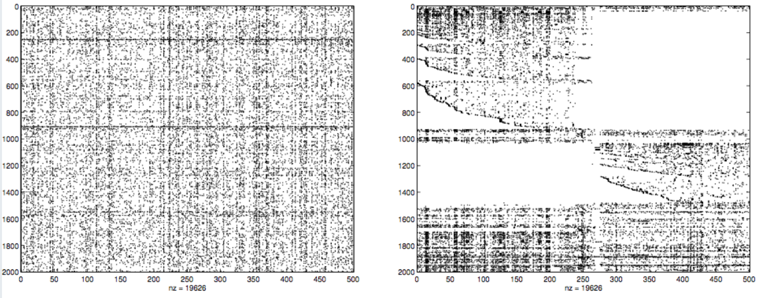
Query-vector Matrix. Let Q be a query log containing queries $q_1, q_2, \ldots, q_m$. Let $D_i = d_{i1}, d_{i2}, \ldots, d_{in_i}$ be the set of documents returned, by a reference search engine, as results to query q_i. $M_{ij} = 1$ if and only if document d_j is in the result set of query q_i (0 if d_j is not a match for q_i).

Co-clustering the Query-vector Matrix

The Idea

Reorder rows and columns of the matrix to obtain dense blocks of 1's

Example



A Nice By-Product of Co-Clustering

The PCAP ($\widehat{P}$) Matrix

Co-clustering produces a matrix we called $\widehat{P}$ representing how rows and columns are cohesive in each cluster

Using $\widehat{P}$

- A query q is scored against each query cluster using a "traditional" ranking score to obtain $r_q(qc_i)$
- The contribution of $\widehat{P}$ for a document cluster dc_j is given by

$$r_q(dc_j) = \sum_i r_q(qc_i) \cdot \widehat{P}(i,j)$$

An Example

Suppose we score the query-clusters respectively 0.2, 0.8 and 0, for a given query q. We compute the vector $r_q(dc_i)$ by multiplying the matrix PCAP by $r_q(qc_i)$, and we will rank the collections dc3, dc1, dc2, dc5, dc4 in this order.

PCAP	dc1	dc2	dc3	dc4	dc5	$r_q(qc_i)$
qc1		0.5	0.8	0.1		0.2
qc2	0.3		0.2		0.1	0.8
qc3	0.1	0.5	0.8			0

$$
\begin{aligned}
r_q(dc_1) &= 0 + 0.3 \times 0.8 + 0 = 0.24 \\
r_q(dc_2) &= 0.5 \times 0.2 + 0 + 0 = 0.10 \\
r_q(dc_3) &= 0.8 \times 0.2 + 0.2 \times 0.8 + 0 = 0.32 \\
r_q(dc_4) &= 0.1 \times 0.2 + 0 + 0 = 0.02 \\
r_q(dc_5) &= 0 + 0.1 \times 0.8 + 0 = 0.08
\end{aligned}
$$

Collection Prioritization

- Collections are ranked w.r.t. a query q
- q is broadcasted along with the ranked list of servers
- The most promising core will receive a query tagged with top priority, equal to 1.
- The other cores c will receive a query q tagged with linearly decreasing priority $p_{q,c}$ (down to $1/N$, with N cores).

Thresholding Strategies

At time t, a core c with current load $l_{c,t}$ will serve the query q if:

$$l_{c,t} \times p_{q,c} < L$$

where L is a load threshold that represents the computing power available to the system.

Incremental Caching

The Idea

- Queries may not be answered by all servers
- Use a *prioritization-aware* caching policy keeping track of what servers are missing from the list of servers for each cached query
- If a query is in cache check if its list of server is complete
- If not, forward the query only to those servers that did not previously answer

Test Collection

The WBR99 test collection

d	5,939,061 documents taking (uncompressed) 22 GB
t	2,700,000 unique terms
t'	74,767 unique terms in queries
tq	494,113 (190,057 unique) queries in the training set
$q1$	194,200 queries in the main test set (first week - TodoBR)

Evaluation Metric

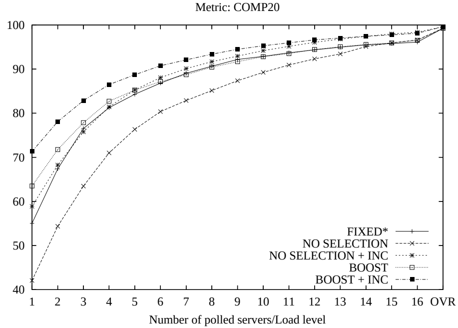
Competitive Similarity

The *competitive similarity at N*, $COMP_N(q)$, measures the relative *quality* of results coming from collection selection with respect to the best results from the central index

Results

Parameters

- Cache Size: 32k results
- Policy: Incremental LRU

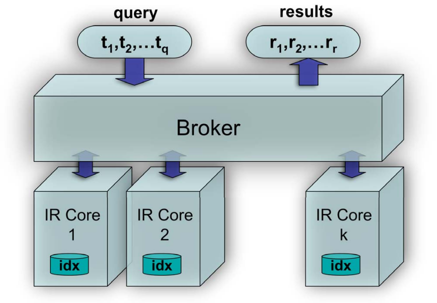


Overall Considerations

- We retrieve more than 1/3 of the most relevant results that a full index would return, by querying only the first server returned by our selection function
- Use the instant load at each server for driving query routing
- We can reach a competitive similarity of about 2/3, with a computing load of 10%, i.e. a server answers no more than 100 queries out of every 1000.
- A system, with a slightly higher load (25%), can reach a whooping 80% competitive similarity w.r.t. a centralized global index.
- More info in [Puppin *et al.*, 2009]

Term Partitioning

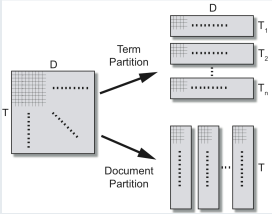
Instead of dividing documents and then separately index them, index documents and split the index along dictionary partitions.



Smart Term Partitioning

What do we mean with the term "*Smart*"?

We want to find a "*Smart*" way to partition the term-document matrix to enhance performance of Term Partitioned IR systems



We want to allow TP systems to answer queries using **few servers** per query (enhancing overall system's capacity) and by **spreading queries** to all the available servers (balancing the load).

Definitions

- q is forwarded to $H_\lambda(Q)$ servers
  - H_λ is the set of servers containing postings lists for some terms of the query according to the partitioning λ of the lexicon

Definitions

- q is forwarded to $H_\lambda(Q)$ servers
  - H_λ is the set of servers containing postings lists for some terms of the query according to the partitioning λ of the lexicon
- Given the pair $(t, l_t) \in \mathcal{I}$, where t is a term of the lexicon and l_t is the length of its postings list, we will use the following symbols:
  - $T_{disk}(|l_t|)$: time to transfer from disk the postings list l_t
  - $T_{compute}(|l_t|)$: time spent on the postings list l_t
  - $T_{overhead}$: CPU time spent by a server in network I/O

Definitions

- q is forwarded to $H_\lambda(Q)$ servers
  - H_λ is the set of servers containing postings lists for some terms of the query according to the partitioning λ of the lexicon
- Given the pair $(t, l_t) \in \mathcal{I}$, where t is a term of the lexicon and l_t is the length of its postings list, we will use the following symbols:
  - $T_{disk}(|l_t|)$: time to transfer from disk the postings list l_t
  - $T_{compute}(|l_t|)$: time spent on the postings list l_t
  - $T_{overhead}$: CPU time spent by a server in network I/O
- Let Q_λ^j be the subsets of the terms in Q assigned to the server j according to the partitioning λ:

$$T_\lambda^j(Q) = T_{overhead} + \sum_{t \in Q_\lambda^j} (T_{disk}(|l_t|) + T_{compute}(|l_t|))$$

Working Hypothesis

Completion Time of queries in Φ

$$\widehat{L}_\lambda(\Phi) = \max_j \sum_{Q \in \Phi} T_\lambda^j(Q)$$

In term-partitioned WSE with a partitioning function λ the following two hypothesis hold

Throughput

$O\left(|\Phi| / \widehat{L}_\lambda\right)$

Query Latency

$O\left(\sum_{Q \in \Phi} H_\lambda(Q) / |\Phi|,\right)$

The Term Assignment Problem

The Term-Assignment Problem. *Given a weight* α, $0 \leq \alpha \leq 1$, *a query stream* Φ, *the Term-Assignment Problem asks for finding the partitioning* λ *which minimizes*

$$\Omega_\lambda(\Phi) = \alpha \cdot \frac{\overline{\omega}_\lambda(\Phi)}{N_\omega} + (1-\alpha) \cdot \frac{\widehat{L}_\lambda(\Phi)}{N_L}$$

where N_ω *and* N_L *are normalization constants.*

The Term Assignment Problem

The Term-Assignment Problem. *Given a weight* α, $0 \leq \alpha \leq 1$, *a query stream* Φ, *the Term-Assignment Problem asks for finding the partitioning* λ *which minimizes*

$$\Omega_\lambda(\Phi) = \alpha \cdot \frac{\overline{\omega}_\lambda(\Phi)}{N_\omega} + (1-\alpha) \cdot \frac{\widehat{L}_\lambda(\Phi)}{N_L}$$

where N_ω *and* N_L *are normalization constants.*

The Idea

The first term is related to *query latency*, the second to the *throughput*.

Query Log Information

- The term partitioning problem has been stated in terms of the query stream Φ.
- Information about future queries are obviously unavailable at partitioning time.
- We can exploit the presence of power law in query logs to extract frequently occurring patterns of terms within queries.
- The idea is to assign frequently co-occurring terms to the same partition.
- Intuitively both $\overline{\omega}$ and $\widehat{L}$ can be optimized by taking into consideration conjunctions of terms. In fact, by assigning to the same partition terms that often co-occur together we reduce both the average width and the overhead due to extra communications.

Experimental Settings

Query Logs Used

Query log	queries	terms	query len.	date
TodoBR	22,589,568	959,833	3.433	2001
AltaVista	7,175,648	895,792	2.507	Summer 2001

- Queries are transformed in lower case
- Query logs were split in $2/3$ for training ($\Phi_{training}$) $1/3$ for testing (Φ_{test})
- We validated our approach by simulating a broker and assuming constant times for T_{disk}, $T_{compute}$, and $T_{overhead}$ disregarding the lengths of the posting lists.
- We considered a partitioning of the index among $p = 8$ servers.

Width of queries

Percentages of queries as a function of the number of servers involved in their processing

Servers	Baseline Cases: random	Baseline Cases: bin packing	Term Assignment $\alpha = 0.9$
$\Phi_{test} = TodoBR$			
1	28	28	50
2	31	30	20
3	17	17	14
> 3	24	25	16
$\Phi_{test} = AltaVista$			
1	29	29	41
2	39	39	38
3	21	21	16
> 3	11	11	5

More info

In [Lucchese *et al.*, 2007] many results have been shown:

- Comparison with simple bin-packing
- Load Balancing
- Term Replication

What's still missing

Testing of "*actual*" performance gains (hopefully) on a real term partitioning search engine system.

Lesson Learned

Using history allows us to...

- Improve both Document and Term Partitioning based search engines

Lesson Learned

Using history allows us to...

- Improve both Document and Term Partitioning based search engines
- Query Vector Model boosts a scheme called Collection Prioritization

Lesson Learned

Using history allows us to...

- Improve both Document and Term Partitioning based search engines
- Query Vector Model boosts a scheme called Collection Prioritization
    - Basically we exploit the association of past submitted queries with returned results

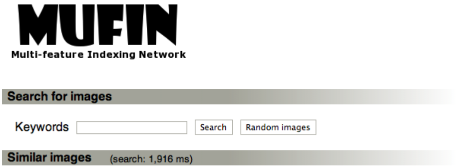
Lesson Learned

Using history allows us to...

- Improve both Document and Term Partitioning based search engines
- Query Vector Model boosts a scheme called Collection Prioritization
  - Basically we exploit the association of past submitted queries with returned results
- The Term Assignment Problem exploit co-occurrence of terms within queries submitted in the past to compute an optimized assignment of terms to partitions

Outline

A Multimedia Retrieval System



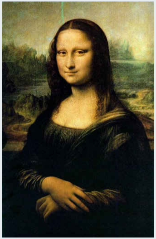
Scalability Issues in Multimedia Retrieval

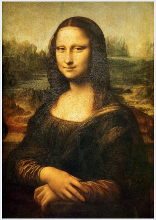
- Traditional (Yahoo!-like) multimedia retrieval is based on textual meta-information devised from the context in which multimedia elements appear.
- Content Based Image Retrieval (CBIR) suffer from scalability issues like:
  - Visual descriptor are expensive to obtain
  - Metrics to compute similarity are fast, yet not fast enough.
- Caching for CBIR can be a viable approach

Caching in CBIR

The Main Issue

Queries are by-example people might look for the same image even if they are submitting different images.

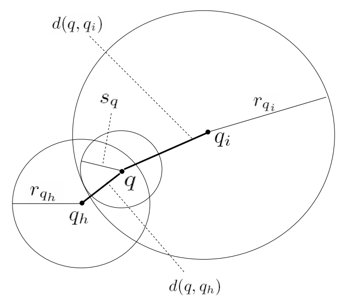




The Need for Similarity Caching

A possible Solution: QCache

When a new query q is submitted, try to retrieve the result set of the closest queries (q_i, q_h in the example) in cache. More details in [Falchi *et al.*, 2008]



The Curse of Lacking Data

CBIR system logs are not available

To generate a realistic log we must take into account:

- The distribution of topic popularity in the log is similar to the one found in text-based query logs [van Zwol, 2007]
- About 8% of the images in the web are near-duplicates [Foo *et al.*, 2007]

[1] http://cophir.isti.cnr.it

The Curse of Lacking Data

CBIR system logs are not available

To generate a realistic log we must take into account:

- The distribution of topic popularity in the log is similar to the one found in text-based query logs [van Zwol, 2007]
- About 8% of the images in the web are near-duplicates [Foo *et al.*, 2007]

Steps to Generate our CBIR Log

- We took CoPhIR[1] and we observed that image popularity in pictures follows a power-law

[1] http://cophir.isti.cnr.it

The Curse of Lacking Data

CBIR system logs are not available

To generate a realistic log we must take into account:

- The distribution of topic popularity in the log is similar to the one found in text-based query logs [van Zwol, 2007]
- About 8% of the images in the web are near-duplicates [Foo *et al.*, 2007]

Steps to Generate our CBIR Log

- We took CoPhIR[1] and we observed that image popularity in pictures follows a power-law
- We injected 8% of duplicate images

[1] http://cophir.isti.cnr.it

The Curse of Lacking Data

CBIR system logs are not available

To generate a realistic log we must take into account:

- The distribution of topic popularity in the log is similar to the one found in text-based query logs [van Zwol, 2007]
- About 8% of the images in the web are near-duplicates [Foo *et al.*, 2007]

Steps to Generate our CBIR Log

- We took CoPhIR[1] and we observed that image popularity in pictures follows a power-law
- We injected 8% of duplicate images
- We sampled $100,000$ images according to their popularity as representative queries

[1] http://cophir.isti.cnr.it

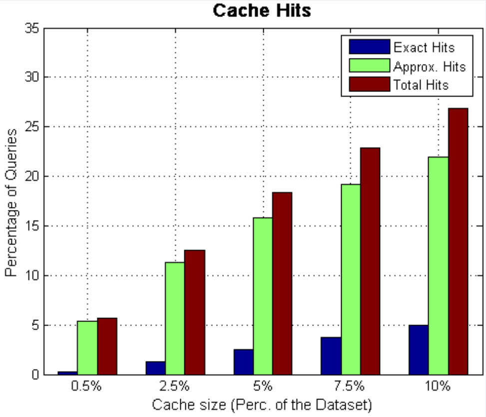
Test Settings

- 1M images from the CoPhIR[2] collection
- The log synthesized as explained before
- An index over the 1M images of the CoPhIR collection built using MTree[3]
- QCache the cache system following the approximate caching strategy depicted above

[2] http://cophir.isti.cnr.it
[3] http://lsd.fi.muni.cz/trac/mtree/

Results

Hit Ratio



Lesson Learned

Using history allows us to...

- State that QCache, an approximate caching policy, is worthwhile

Lesson Learned

Using history allows us to...

- State that QCache, an approximate caching policy, is worthwhile
  - approximate, here, means that we search for similar previously submitted queries within cache entries

Outline

Conclusion

My Two Cents

- Using query logs is very important for improving the efficiency of Search Engine systems

More to come... Stay Tuned!

Fabrizio Silvestri, *Mining Query Logs: Turning Search Usage Data into Knowledge* , Foundations and Trends in Information Retrieval. 2009. To Appear.

Conclusion

My Two Cents

- Using query logs is very important for improving the efficiency of Search Engine systems
- Uses different from pure caching has been shown to be effective

More to come... Stay Tuned!

Fabrizio Silvestri, *Mining Query Logs: Turning Search Usage Data into Knowledge* , Foundations and Trends in Information Retrieval. 2009. To Appear.

QUESTIONS????

References

- [Baeza-Yates *et al.*, 2007a] R. Baeza-Yates, A. Gionis, F. Junqueira, V. Murdock, V. Plachouras, and F. Silvestri, *The impact of caching on search engines*, Proceedings of the ACM SIGIR 2007 (New York, NY, USA), ACM, 2007, pp. 183–190.

- [Baeza-Yates *et al.*, 2007b] Ricardo A. Baeza-Yates, Flavio Junqueira, Vassilis Plachouras, and Hans Friedrich Witschel, *Admission policies for caches of search engine results*, SPIRE (Nivio Ziviani and Ricardo A. Baeza-Yates, eds.), Lecture Notes in Computer Science, vol. 4726, Springer, 2007, pp. 74–85.

- [Beitzel *et al.*, 2007] Steven M. Beitzel, Eric C. Jensen, Abdur Chowdhury, Ophir Frieder, and David Grossman, *Temporal analysis of a very large topically categorized web query log*, J. Am. Soc. Inf. Sci. Technol. **58** (2007), no. 2, 166–178.

References

- [Markatos, 2000] Markatos E.P., *On caching search engine query results*, Computer Communications **24** (1 February 2000), 137–143(7).
- [Fagni *et al.*, 2006] Tiziano Fagni, Raffaele Perego, Fabrizio Silvestri, and Salvatore Orlando, *Boosting the performance of web search engines: Caching and prefetching query results by exploiting historical usage data*, ACM Trans. Inf. Syst. **24** (2006), no. 1, 51–78.
- [Falchi *et al.*, 2008] Fabrizio Falchi, Claudio Lucchese, Salvatore Orlando, Raffaele Perego, and Fausto Rabitti, *A metric cache for similarity search*, Proceedings of the 6th Workshop on Large-Scale Distributed Systems for Information Retrieval (LSDS-IR'08), October 30th 2008.

References

[Foo *et al.*, 2007] J. J. Foo, J. Zobel, R. Sinha, and S. M. M. Tahaghoghi, *Detection of near-duplicate images for web search*, Proceedings of ACM CIVR '07 (New York, NY, USA), ACM, 2007, pp. 557–564.

[Lempel and Moran, 2003] Ronny Lempel and Shlomo Moran, *Predictive caching and prefetching of query results in search engines*, WWW '03: Proceedings of the 12th international conference on World Wide Web (New York, NY, USA), ACM, 2003, pp. 19–28.

[Lucchese *et al.*, 2007] C Lucchese, S. Orlando, R. Perego, and F. Silvestri, *Mining query logs to optimize index partitioning in parallel web search engines*, Infoscale, 2007, p. 43.

References

- [Puppin *et al.*, 2009] D. Puppin, F. Silvestri, R. Perego, and R. Baeza-Yates, *Tuning the capacity of search engines: Load-driven routing and incremental caching to reduce and balance the load*, to Appear in ACM TOIS (2009).

- [Correia Saraiva *et al.*, 2001] Paricia Correia Saraiva, Edleno Silva de Moura, Novio Ziviani, Wagner Meira, Rodrigo Fonseca, and Berthier Ribeiro-Neto, *Rank-preserving two-level caching for scalable search engines*, SIGIR '01: Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval (New York, NY, USA), ACM, 2001, pp. 51–58.

- [van Zwol, 2007] R. van Zwol, *Flickr: Who is looking?*, Proceedings of WI 2007, November 2007.