Optimizing Result Prefetching in Web Search Engines with Segmented Indices

Ronny Lempel  Shlomo Moran
Technion, Israel Institute of Technology
Web Search Engines

• Index billions of documents.
• Make use of distributed storage.
• Their main task: Answer user’s queries – millions per day.
• Issue addressed: Optimizing prefetching policy for answering Active Queries.
Talk Outline

1. **Defining** Active Queries and Active Search Session.
2. **Modeling** the cost of serving Active Queries and Active Search Sessions.
3. **Optimizing** cost of Active Search Session.
4. Possible implementations
Part 1
Active Queries
Active Search Sessions
Query execution: Parties involved (in typical large search engines)

Documents are assigned to segments randomly, uniformly and independently (local inverted index organization)
Search Session: User’s point of view

Submit an *initial* query, then wait...

... eventually receive back a result page, (usually, the top 10 documents for the query)
Search Session (cont.)

...possibly submit a **follow-up** query, requesting additional results

...Receive the following **result page**, and so on until no further follow-up query is asked
Search Session: QI’s point of view

Initial query received...

Are the results stored in the cache?

Case 1. 1st result page (top 10 results) is cached: return to user.
Case 2. 1st result page is not cached. This is an Active Query, needs more work:
QI’s work on Active Query

Ask the **Segmented Index** for the top $n$ results ($n$ to be determined)
QI’s work on Active Query (cont.)

Each of the $m$ segments returns its top $Z$ results ($z=z(m,n)$ to be discussed later)
QI’s work on Active Query (cont.)

QI selects top $n$ (out of $mz$) results, for preparing $r=n/10$ result page(s)

Returns 1st page (10 results) to user

Stores the other $r-1$ pages (if $r>1$) in the cache
Active Search Session:
A (suffix of a) Search Session which starts with Active Query

Goal: Optimize QI’s policy for serving a “typical” Active Search Session
Part 2

Modeling cost of Active Search Session (of a “typical” user)
Cost of Active Search Session depends on:

A. Architectural constraints and query specific parameters (not controlled by the QI)

1. C – the number of relevant documents held in each segment (could be in the millions).
2. m - Number of index segments.
3. Other properties of the search engine.
Cost of Active Search Session depends on:

B. Implementation dependent parameters (controlled by the QI):

1. \( n = 10r \), the number of results the QI prepares.
2. \( z = z_q(n,m) = \# \) of results each of the \( m \) segments returns, so that with probability \( q \), these \( mz \) results contain the top \( n \) results.
The QI needs to decide $n$ -

The Prefetching Dilemma:

1. Preparing just the single required result page is cheap, but follow up queries will require additional Active Query executions.

2. Prefetching several result pages is expensive, but save Active Query executions should the user request the prefetched pages while they are still cached.

Need to model User Behaviour
Search engine users - some statistics:

(based on 3 published analyses of search engine query logs):

A typical user views relatively few result pages

- 1st Pages (containing the top-10 results of queries) account for at least 58% of all result pages viewed by users.
- Pages in ranks 1-3 (containing the top-30 results) account for at least 88% of all page views.
Search Engine users: Views of Result Pages 1-6

Analysis of an AltaVista query log containing ~7.55 million result page views from September, 2001:

About 4.8 million views (63.5%) are for the first page of results (the top-10 results).
Search Engine users: Views of Result Pages 6-20

![Bar Chart: Views of Pages 6-20](chart.png)

- Views on Result Pages 6-20
- X-axis: Result Page Number
- Y-axis: Views
- Chart shows decreasing views from Page 6 to Page 20.

---

VLDB 2002
Modeling Search Sessions

• Result pages are always viewed in their natural order.

• The number of result pages that are viewed per session is a geometric random variable with parameter $p$.

$\Rightarrow$ The probability that a user will view precisely result pages $1,\ldots,k$ of any query in a session is $(1-p)p^{k-1}$.
Executing an Active Query: Work at segments

• Each of the $m$ segments selects the top $z$ results. Assuming documents are evenly distributed, all segments perform roughly the same amount of work, which depends on:
  – The number of documents relevant to the query (query’s breadth).
  – the number $z$ of results that each segment returns to the QI.
Executing an Active Query: Work at QI

• The QI’s work is dominated by receiving the $m_z$ results from the segments, and merging them to get the top $n=10r$ results.

• The cache space required depends on $n=10r$, the number of results that are prepared.

• A single round of communication between the QI and the segments is performed.
W(r): The expected cost of Active Search Session

\[ W = \sum_{i=1}^{\infty} \Pr(\text{executing the } i^{th} \text{ Active Query}) \times \text{(cost of executing each Active Query)} \]

When preparing \( r \) pages, this cost is:

\[ W(r) = ar + \left[ \frac{(b+cz_q(r,m)+dr)}{(1-pr)} \right] \]

Note: For each topic, there is a fixed optimal value \( r_{opt} \) which optimizes the expected work.
Part 3

Optimizing cost of typical Active Search Session.
Optimizing $W(r)$: calculating $r_{opt}$

• The paper presents a polynomial time algorithm that, given a query-topic $T$, determines $r_{opt}$, the number of result pages to prepare per query execution so as to minimize the work function $W(r)$.

• In addition, a (simpler) polynomial time algorithm which, given a parameter $\varepsilon$, outputs a value $r_{\varepsilon}$ for which $W(r_{opt}) / W(r_{\varepsilon}) \geq 1 - \varepsilon$, and is applicable to any (monotonic) cost function, assuming the number of viewed pages is geometric RV.
Optimizing $W(r)$:
Calculating $z_q(r,m)$

Problem demonstration:
m = 5, r = 1 (hence $n = 10$)

How many results should the QI retrieve from each segment?

Fetching the top three results from each of the five segments fails to collect all top 10 results!
Calculating $z_q(r,m)$

- **Notations:**
  1. $m$ – number of segments.
  2. $n$ – number of requested results ($n=10r$).

- In order to collect the top-$n$ results with probability 1, the QI must retrieve $n$ results from each segment.

- In order to have any chance of collecting the top-$n$ results, the QI must retrieve at least $\lceil n/m \rceil$ results from each segment.
Calculating $z_q(r,m)$

- The paper presents a polynomial time DP algorithm to calculate $z_q(r,m)$, the minimal number of results that the QI can retrieve from each segment and still obtain the top-$n$ ($n=10r$) results with probability $q$.
- Applicable to any search engine that uses an $m$–way locally segmented index where documents are distributed uniformly and independently.
Sample values of $z_q(r,m)$, $q=0.99$

![Graph showing sample values of $z_q(r,m)$ for different numbers of segments.](image-url)
Behavior of $W(r) = ar + \frac{b+cz_q(r,m)+dr}{1-pr}$

- $q=0.99$, $m=25$
- $p=0.5$
- Query breadth: $2^{13}$ results per segment
- $r_{opt} = 6$
Part 4

Possible implementations
Implementation of prefetching policy

1. A preprocessing stage: calculate and store relevant data.
2. Query execution stage: determine the parameters $n, z$ by the data stored and the query topic.
Prefetching Framework: Preprocessing Stage

- Determine the work function $W(r)$ that describes the resources that the engine consumes during a Search Session.

- For a wide range of query breadths, calculate $r_{opt}$ and the corresponding values of $z_q(r_{opt}, m)$.

- Load tables with the values of $r_{opt}$ and $z_q(r_{opt}, m)$ into the QI and each of the index segments.
Prefetching Framework: During Query Execution

• Centralized approach:
  
  – The QI estimates the query’s breadth using global term statistics.
  
  – Looks up the corresponding values of $r_{opt}$ and $z_q(r_{opt}, m)$.
  
  – Queries each segment for $z_q(r_{opt}, m)$ results and prepares $r_{opt}$ result pages.
Prefetching Framework: During Query Execution

• Distributed approach:
  – Each segment estimates the query’s breadth independently by considering the number of query matches it contains.

  Looks up the corresponding $z_q(r_{opt}, m)$, and returns this number of results to the QI.
Prefetching Framework: During Query Execution

• Distributed approach (cont.):
  – The QI (conservatively) estimates the query’s breadth from the number of returned results, and prepares the required number of result pages.
Summary and Future Work

• This work aims at minimizing the expected computational load of an Active Search Session.

• Some limitation: the model is insensitive to popularity of queries; the replacement policy of the query result cache is not considered.

• Future work: integrate result prefetching with cache replacement policies. Implement (or simulate) prefetching policies on actual search engines.