Query-Driven Document Partitioning and Collection Selection

Diego Puppin and Fabrizio Silvestri

Institute for Information Science and Technologies
ISTI-CNR
Pisa, Italy

July 10 2006
Outline

1. Introduction
2. The Query-vector Model
3. Experiments
4. Conclusions
Distributed Search Engines

The Web is growing larger and we need to manage more pages

Data are partitioned on several servers with many goals in mind

- Load-balancing
- Increased through-put
- Higher quality results
- Load-reduction
### Example

<table>
<thead>
<tr>
<th>term</th>
<th>doc1</th>
<th>doc2</th>
<th>doc3</th>
<th>doc4</th>
<th>doc5</th>
<th>doc6</th>
</tr>
</thead>
<tbody>
<tr>
<td>term1</td>
<td>0.5</td>
<td>0.8</td>
<td>0.1</td>
<td>0.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>term2</td>
<td>0.3</td>
<td>0.2</td>
<td>0.1</td>
<td>0.8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>term3</td>
<td>0.1</td>
<td>0.5</td>
<td>0.8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>term4</td>
<td>0.2</td>
<td>0.5</td>
<td>0.2</td>
<td></td>
<td></td>
<td>0.5</td>
</tr>
<tr>
<td>term5</td>
<td>0.1</td>
<td>0.1</td>
<td></td>
<td>0.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>term6</td>
<td></td>
<td></td>
<td></td>
<td>0.3</td>
<td>0.6</td>
<td>0.3</td>
</tr>
</tbody>
</table>
### Example: Document Partitioned

<table>
<thead>
<tr>
<th></th>
<th>doc1</th>
<th>doc2</th>
<th>doc3</th>
<th>doc4</th>
<th>doc5</th>
<th>doc6</th>
</tr>
</thead>
<tbody>
<tr>
<td>term1</td>
<td></td>
<td>0.5</td>
<td>0.8</td>
<td>0.1</td>
<td></td>
<td>0.2</td>
</tr>
<tr>
<td>term2</td>
<td>0.3</td>
<td></td>
<td>0.2</td>
<td></td>
<td>0.1</td>
<td>0.8</td>
</tr>
<tr>
<td>term3</td>
<td>0.1</td>
<td>0.5</td>
<td></td>
<td>0.8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>term4</td>
<td>0.2</td>
<td>0.5</td>
<td></td>
<td>0.2</td>
<td></td>
<td>0.5</td>
</tr>
<tr>
<td>term5</td>
<td></td>
<td></td>
<td>0.1</td>
<td></td>
<td>0.2</td>
<td>0.5</td>
</tr>
<tr>
<td>term6</td>
<td>0.1</td>
<td></td>
<td></td>
<td>0.3</td>
<td>0.6</td>
<td>0.3</td>
</tr>
</tbody>
</table>
### Example: Term Partitioned

<table>
<thead>
<tr>
<th></th>
<th>doc1</th>
<th>doc2</th>
<th>doc3</th>
<th>doc4</th>
<th>doc5</th>
<th>doc6</th>
</tr>
</thead>
<tbody>
<tr>
<td>term1</td>
<td></td>
<td>0.5</td>
<td>0.8</td>
<td>0.1</td>
<td></td>
<td>0.2</td>
</tr>
<tr>
<td>term2</td>
<td>0.3</td>
<td></td>
<td>0.2</td>
<td></td>
<td>0.1</td>
<td>0.8</td>
</tr>
<tr>
<td>term3</td>
<td></td>
<td>0.1</td>
<td>0.5</td>
<td>0.8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>term4</td>
<td>0.2</td>
<td>0.5</td>
<td></td>
<td>0.2</td>
<td></td>
<td>0.5</td>
</tr>
<tr>
<td>term5</td>
<td></td>
<td>0.1</td>
<td>0.1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>term6</td>
<td></td>
<td></td>
<td></td>
<td>0.3</td>
<td>0.6</td>
<td>0.3</td>
</tr>
</tbody>
</table>
Term-partitioned Index

- Terms are assigned to servers
- Queries are submitted to servers holding the relevant terms
- Only a subset of servers is queried
- Results from each server are intersected/merged and ranked
- Problem of load-balancing, very hard to assign terms
  - Some recent works about this
- Can reduce the overall system load
Introduction

Document-partitioned Index

- Documents are assigned to servers
- A query can be submitted to each cluster, to improve throughput
- ... OR ... to reduce load, only to selected servers
- We must choose the “good servers” in advance
- Problem of partitioning and collection selection
- Back to the problems of heterogeneous collections (CORI etc.)
Several Approaches to Partitioning and Selection

Document partitioning:
- Document clustering with k-means
- Semantic cataloguing with ontologies
- Random/round robin

Collection Selection:
- CORI
- Random
- All collections are queried
- Online sampling
Documents are partitioned randomly
Queries are sent to all servers
  Load-balancing
Results from all servers are merged/ranked
Outline

1. Introduction
2. The Query-vector Model
3. Experiments
4. Conclusions
Two Birds with One Stone

- We are trying to make clusters of documents that answer to similar query
- We are also trying to clusters queries that recall similar documents
- We have to co-cluster [Dhillon 2003] the query-document matrix
- Very fast algorithm (much faster than k-means)
Coclustering Example

\[
p(X, Y) = \begin{bmatrix}
.05 & .05 & .05 & 0 & 0 & 0 \\
.05 & .05 & .05 & 0 & 0 & 0 \\
0 & 0 & 0 & .05 & .05 & .05 \\
0 & 0 & 0 & .05 & .05 & .05 \\
.04 & .04 & 0 & .04 & .04 & .04 \\
.04 & .04 & .04 & 0 & .04 & .04 \\
\end{bmatrix}
\]

\[
p(\hat{X}, \hat{Y}) = \begin{bmatrix}
.3 & 0 \\
0 & .3 \\
.2 & .2 \\
\end{bmatrix}
\]

Rows and columns are shuffled to minimize loss of information.
Our Approach

- For every training query, we store the first 100 results of a reference search engine (centralized index)
- We create a query-document matrix, entries proportional to rank
- We co-cluster to put 1’s and 0’s together (actually, float numbers)
- We create $N$ document clusters and $M$ query clusters
- The process minimizes the loss of information between the original and the clustered matrix

$$\hat{P}(qc_a, dc_b) = \sum_{i \in qc_b} \sum_{j \in dc_a} r_{ij}$$
### Query-vector Representation

For each query, we store the Top-100 results with rank

<table>
<thead>
<tr>
<th>Query/Doc</th>
<th>d1</th>
<th>d2</th>
<th>d3</th>
<th>d4</th>
<th>d5</th>
<th>d6</th>
<th>...</th>
<th>dn</th>
</tr>
</thead>
<tbody>
<tr>
<td>q1</td>
<td>-</td>
<td>0.5</td>
<td>0.8</td>
<td>0.4</td>
<td>-</td>
<td>0.1</td>
<td>...</td>
<td>-</td>
</tr>
<tr>
<td>q2</td>
<td>0.3</td>
<td>-</td>
<td>0.2</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.1</td>
</tr>
<tr>
<td>q3</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>q4</td>
<td>-</td>
<td>0.4</td>
<td>-</td>
<td>0.2</td>
<td>-</td>
<td>0.5</td>
<td>...</td>
<td>0.3</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>qm</td>
<td>0.1</td>
<td>0.5</td>
<td>0.8</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>...</td>
<td>-</td>
</tr>
</tbody>
</table>

We may have **empty** columns (documents never recalled, d5) and empty rows (queries with no results, q3). They are removed before co-clustering. About 52% of documents are recalled by NO query - we can put them in an **overflow** cluster.
What Happens?

- We put together 1’s and 0’s
- We create highly correlated groups of documents and queries
- Each entry in the co-clustered matrix represent the *affinity* of a document cluster to a query cluster
We create big *query dictionaries* by chaining together all the queries from one query-cluster.

We index the dictionaries as documents.

For a new query \( q \), we choose the best query-clusters with TF.IDF:

- For each query-cluster \( qc_i \), we get a rank \( r_q(qc_i) \).

We can compute the rank of each document-cluster:

\[
r_q(dc_j) = \sum_{i} r_q(qc_i) \times \hat{P}(i, j)
\]

The overflow IR core is always queried as the last one.
PCAP Example

<table>
<thead>
<tr>
<th></th>
<th>dc1</th>
<th>dc2</th>
<th>dc3</th>
<th>dc4</th>
<th>dc5</th>
<th>Rank for q</th>
</tr>
</thead>
<tbody>
<tr>
<td>qc1</td>
<td>0.5</td>
<td>0.8</td>
<td>0.1</td>
<td></td>
<td></td>
<td>0.2</td>
</tr>
<tr>
<td>qc2</td>
<td>0.3</td>
<td>0.2</td>
<td></td>
<td>0.1</td>
<td></td>
<td>0.8</td>
</tr>
<tr>
<td>qc3</td>
<td>0.1</td>
<td>0.5</td>
<td>0.8</td>
<td></td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

Query \( q \) ranks the \( qc \) respectively 0.2, 0.8 and 0.

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

Clusters will be chosen in the order dc3, dc1, dc2, dc5, dc4.
### Data Statistics

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>dc</td>
<td>no. of document clusters</td>
<td>16 + 1</td>
</tr>
<tr>
<td>qc</td>
<td>no. of query clusters</td>
<td>128</td>
</tr>
<tr>
<td>d</td>
<td>no. of documents</td>
<td>5,939,061</td>
</tr>
<tr>
<td></td>
<td>total size</td>
<td>22 GB</td>
</tr>
<tr>
<td>t</td>
<td>no. of unique terms</td>
<td>2,700,000</td>
</tr>
<tr>
<td>t'</td>
<td>no. of unique terms in the query dictionary</td>
<td>74,767</td>
</tr>
<tr>
<td>tq</td>
<td>no. of unique queries in the training set</td>
<td>190,057</td>
</tr>
<tr>
<td>q1</td>
<td>no. of queries in the first test set</td>
<td>194,200</td>
</tr>
<tr>
<td>q2</td>
<td>no. of queries in the second test set</td>
<td>189,848</td>
</tr>
<tr>
<td>ed</td>
<td>empty (not recalled) documents</td>
<td>3,128,366</td>
</tr>
</tbody>
</table>

**Table:** Statistics about collection representation. Data and query-logs from WBR99.
Benchmarks

Partitions based on document contents:
- Random allocation
- Clusters with shingles
  - Signature of 64 permutations
- URL sorting

Partitions based on query-vector representation:
- Clustering with k-means
- Co-clustering (*)

(*) We could use PCAP in this case!
### Precision with one cluster

<table>
<thead>
<tr>
<th>Method</th>
<th>Precision at 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>random allocation (CORI)</td>
<td>0.3</td>
</tr>
<tr>
<td>clustering with shingles (CORI)</td>
<td>0.56</td>
</tr>
<tr>
<td>URL sorting (CORI)</td>
<td>0.94</td>
</tr>
<tr>
<td>clustering with k-means on query-vectors (CORI)</td>
<td>1.47</td>
</tr>
<tr>
<td>co-clustering (CORI)</td>
<td>1.57</td>
</tr>
<tr>
<td>co-clustering (PCAP)</td>
<td>1.74</td>
</tr>
</tbody>
</table>

**Table:** Precision at 5 on the first cluster.
Impact

- If a given precision is expected, we can use FEWER servers.
- With a given number of servers, we get HIGHER precision.
  - Confirmed with different metrics.
- Smaller load for the IR system, with better results.
- *No load balancing (for now)*
- 50% of pages contribute to 97% precision.
  - We can remove the rest.
Robustness to Topic Drift

Results do not change significantly if we do our test with later queries.

<table>
<thead>
<tr>
<th>Precision at</th>
<th>FOURTH WEEK</th>
<th>FIFTH WEEK</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>1.74</td>
<td>2.30</td>
</tr>
<tr>
<td>10</td>
<td>3.45</td>
<td>4.57</td>
</tr>
<tr>
<td>20</td>
<td>6.93</td>
<td>9.17</td>
</tr>
</tbody>
</table>

Table: Precision at 5 of the PCAP strategy, on the 4th and the 5th week.
CORI representation includes:

- $df_{i,k}$, the number of documents in collection $i$ containing term $k$, which is $O(dc \times t)$ (before compression),
- $cw_i$, the number of different terms in collection $i$, $O(dc)$,
- $cf_k$, the number of resources containing the term $k$, $O(t)$.

Total: $O(dc \times t) + O(dc) + O(t)$ (before compression)

$dc$, number of document clusters (16+1)
$t$, number of distinct terms, 2,700,000
The PCAP representation is composed of:

- the PCAP matrix, with the computed $\hat{p}$, which is $O(dc \times qc)$,
- the index for the query clusters, which can be seen as $n_{i,k}$, the number of occurrences of term $k$ in the query cluster $i$, for each term occurring in the queries — $O(qc \times t')$.

**TOTAL:** $O(dc \times qc) + O(t' \times qc) = 9.4M$ (uncompressed)

**CORI:** $O(dc \times t) + O(dc) + O(t) = 48.6M$ (uncompressed)

$dc$, number of document clusters, 16+1
$qc$, number of query clusters, 128
$t'$, number of distinct terms in the query dictionary, 74,767
$t$, number of distinct terms, 2,700,000
Outline

1. Introduction
2. The Query-vector Model
3. Experiments
4. Conclusions
Conclusions

Can this be done on a real engine?

- Billions of documents, and millions of queries daily
  - How large is the doc/query matrix?
- Is considering ONLY clicked results, better or worse?
  - Not ranking, but clicking
- Is the user happy with a smaller precision (w.r.t. full index)?
  - Is the overall system happier?
- What about load balancing?
  - We never tried
Scalability

- Co-clustering is highly data-parallel...
- but can you reassign documents and shuffle the index?
- Is collection selection fast enough?
- Do we get the results faster, overall?
- Is it worth having a hybrid clustering?
  - We create only few document clusters, and we do round robin inside
Happy Algorithms to You(!)

- New (smaller) document representation as query-vectors
  - 2.7 M terms vs. 190 K queries
  - More effective on clustering (k-means)
  - Helps with the curse of dimensionality
- New partitioning strategy based on co-clustering
  - Very quick running time
- New (smaller) collection representation based on PCAP matrix
  - About 19% in size before compression
- New strategy PCAP for collection selection
  - 10% better than CORI on different metrics
- Removal of 50% of rarely-asked-for documents with minimal loss
  - They contribute only to 3% of recalled documents
Happy Systems to You(?)

- Include click-through data in the reference engine and precision evaluation
- Address load-balancing and overall system performance
- Complete a deeper analysis of the query-vector representation for IR tasks
THANK YOU for coming