Query-Driven Document Partitioning and Collection Selection

Diego Puppin, Fabrizio Silvestri and Domenico Laforenza

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

May 31, 2006
1. Introduction
2. The Query-vector Model
3. Experiments, With Exciting Unpublished Data!
4. Conclusions
Distributed Search Engines

- The Web is growing larger and we need to manage more pages
- Replicated/Distributed Search Engines are a way to tackle
- Two main ways to partition the index
  - Document-partitioned
  - Term-partitioned
- Sometimes with different goals
  - Load-balancing
  - Throughput
  - Load-reduction
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
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 clustering with directories
- Random/round robin

Collection Selection:
- CORI
- Random
- All collections are queried
- Online sampling

Now, we are trying something new!
The Query-vector Model

Outline

1. Introduction

2. The Query-vector Model

3. Experiments, With Exciting Unpublished Data!

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.
Collection Selection using PCAP

- 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>0.8</td>
<td></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.
Outline

1. Introduction
2. The Query-vector Model
3. Experiments, With Exciting Unpublished Data!
4. Conclusions
Data Statistics

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

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

Benchmarks

Partitions based on document contents:

- Random allocation
- Clusters with shingles UNPUBLISHED!!!
  - Signature of 64 permutations
- URL sorting UNPUBLISHED!!!

Partitions based on query-vector representation:

- Clustering with k-means UNPUBLISHED!!!
- Co-clustering (*)

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

<table>
<thead>
<tr>
<th>Method</th>
<th>Precision</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.
Experiments

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></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>1.74</td>
<td>2.30</td>
<td>2.95</td>
<td>3.83</td>
<td>4.85</td>
<td>5.00</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>3.45</td>
<td>4.57</td>
<td>5.84</td>
<td>7.60</td>
<td>9.67</td>
<td>10.00</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>6.93</td>
<td>9.17</td>
<td>11.68</td>
<td>15.15</td>
<td>19.31</td>
<td>20.00</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Precision at</th>
<th>FIFTH WEEK</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>1.73</td>
<td>2.26</td>
<td>2.89</td>
<td>3.76</td>
<td>4.84</td>
<td>5.00</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>3.47</td>
<td>4.51</td>
<td>5.75</td>
<td>7.50</td>
<td>9.66</td>
<td>10.00</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>6.92</td>
<td>9.02</td>
<td>11.47</td>
<td>14.98</td>
<td>19.29</td>
<td>20.00</td>
<td></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:** $O(dc \times qc) + O(t' \times qc) = 9.4\text{M}$ (uncompressed)

**CORI:** $O(dc \times t) + O(dc) + O(t) = 48.6\text{M}$ (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, With Exciting Unpublished Data!
4. Conclusions
Main Contributions

- 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
Next Steps

We would like to:

- include click-through data in the reference engine and precision evaluation;
  - ...if you have them, please share :-)...
- address load-balancing and overall system performance;
- complete a deeper analysis of the query-vector representation for IR tasks;
- compare of document- and term-partitioning.
Acknowledgments

- MIUR CNR Strategic Project L 499/97-2000 (5%)
- NextGrid
- CoreGRID
- ISTI-CNR