Efficient Document Retrieval in Main Memory

Trevor Strohman • W. Bruce Croft
University of Massachusetts Amherst
What if we put everything in memory?
What if we put everything in memory?

Does in-memory processing allow for new efficient retrieval techniques?
Results
<table>
<thead>
<tr>
<th>Method</th>
<th>2005</th>
<th>2006</th>
<th>2007</th>
</tr>
</thead>
<tbody>
<tr>
<td>Doc Sorted</td>
<td>0.6</td>
<td>7x</td>
<td>70%</td>
</tr>
<tr>
<td>Pruning</td>
<td>0.6</td>
<td>7x</td>
<td>57%</td>
</tr>
<tr>
<td>Memory</td>
<td>0.6</td>
<td>7x</td>
<td>66.7%</td>
</tr>
<tr>
<td>Trimming</td>
<td>0.6</td>
<td>7x</td>
<td>70%</td>
</tr>
<tr>
<td>Skips</td>
<td>0.6</td>
<td>7x</td>
<td>97.5%</td>
</tr>
</tbody>
</table>
Exact results only

no static pruning
Büttcher, Carmel

no approximate pruning
Lester: Adaptive Pruning
Moffat/Zobel: Quit and Continue
Components
Everything in memory

Efficient dynamic pruning
Everything in memory
index is memory mapped
8GB of RAM, 7GB index

Efficient dynamic pruning
Everything in memory
index is memory mapped
8GB of RAM, 7GB index

Efficient dynamic pruning
accumulator trimming
inverted list skipping
Dynamic Pruning
accumulator

maximum possible score for any document

maximum possible score for this document

uncertain region

minimum possible score for this document

minimum possible score for any document
6 accumulators, top-3 retrieval
6 accumulators, top-3 retrieval
6 accumulators, top-3 retrieval
6 accumulators, top-3 retrieval
6 accumulators, top-3 retrieval
6 accumulators, top-3 retrieval
6 accumulators, top-3 retrieval
unseen documents
documents that have not yet been seen in inverted lists
trimming
top documents determined

top documents are known, although order is not
top documents determined

top documents are known,
although order is not
top documents determined

top documents are known, although order is not
ranking determined

ranking is known, although exact scores are not
Number of postings processed

Size of accumulator table

Or

And

Refine

Ignore

consider everything

ignore unseen

top documents determined

ranking determined
Number of postings processed

Size of accumulator table

Or

And

Refine

Ignore

consider everything

ignore unseen

top documents determined

ranking determined
Number of postings processed

Size of accumulator table

Or

And

Refine

Ignore

consider everything

ignore unseen

ignore more

top documents determined

ranking determined
<table>
<thead>
<tr>
<th></th>
<th>GOV2 queries/second</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Doc Sorted</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Strohman 2005</strong></td>
<td>0.6</td>
</tr>
<tr>
<td><strong>Pruning</strong></td>
<td>7.5</td>
</tr>
<tr>
<td><strong>Memory</strong></td>
<td>57.8%</td>
</tr>
<tr>
<td><strong>Trimming</strong></td>
<td>66.7</td>
</tr>
<tr>
<td><strong>Skips</strong></td>
<td>97.5</td>
</tr>
<tr>
<td><strong>Strohman/Croft 2007</strong></td>
<td></td>
</tr>
</tbody>
</table>
Skipping
Number of postings processed

Size of accumulator table

Or

And

Refine

Ignore
skipping
lookup table allows fast access to segments of the compressed list
skipping
lookup table allows fast access to segments of the compressed list
Skipping disadvantages

extra code complexity
more data to read and process

Solution

dynamically choose when to skip
if in OR mode:
  **don’t** skip

if in AND mode with *many* accumulators:
  **don’t** skip

if in AND mode with *few* accumulators:
  **do** skip
Results, Part II
everything in memory
efficient dynamic pruning
The chart shows the performance improvements in GOV2 queries/second over the years for different methods and their associated improvements.

- **Doc Sorted (Strohman 2005):** 0.6
- **Pruning (Anh/Moffat 2006):** 7.5
- **Memory (Strohman 2006):** 57.8
- **Trimming (Strohman/Croft 2007):** 66.7
- **Skips (Strohman/Croft 2007):** 97.5

Improvements:
- **7x** increase in Pruning efficiency.
- 70% increase in overall performance.

The chart indicates a significant improvement in efficiency and performance over time.
query terms

GOV2 milliseconds/query

Memory
Trimming
Skips

1
2
3
4
5
6
7
8
The chart shows the GOV2 milliseconds per query for different numbers of query terms. It includes categories for Anh/Moffat, Trimming, and Skips. The x-axis represents the number of query terms, ranging from 1 to 8, while the y-axis indicates the GOV2 milliseconds per query, ranging from 0 to 400. The chart highlights that as the number of query terms increases, the GOV2 milliseconds also increase, particularly significant for 8 query terms.
The graph shows the performance of GOV2 queries/second across different processor core counts, with three categories: Memory, Trimming, and Skips.

- **Processor Cores**: 1, 2, 4
- **GOV2 queries/second**:
  - Memory: 97, 161, 228
  - Trimming: 97, 161, 228
  - Skips: 1x, 1.65x, 2.35x

The performance increases as the number of processor cores increases. For example:
- With 1 core, Memory is 97 queries/second.
- With 2 cores, Memory is 161 queries/second, which is 1.65x the performance of 1 core.
- With 4 cores, Memory is 228 queries/second, which is 2.35x the performance of 1 core.

The graph indicates a significant improvement in performance as the number of cores increases.
In the paper
model for skip length performance
skip length training
accumulator structures
cache effects in skipping
TREC 2006 results
query length distributions
Conclusion
Everything in memory

Efficient dynamic pruning
Everything in memory
it may be worth spending extra for more memory
7x performance improvement
memory bandwidth is important

Efficient dynamic pruning
Everything in memory
it may be worth spending extra for more memory
7x performance improvement
memory bandwidth is important

Efficient dynamic pruning
like static pruning without the risk
70% faster than previous approaches
skipping and trimming work together
important to study the memory-only case
Java-based distributed indexer
C++ and Java retrieval engines

Fall 2007