FROM QUERIES TO TOP-K RESULTS
Outline

- Intro
- Basics of probability and information theory
- Retrieval models
- Retrieval evaluation
- Link analysis
- From queries to top-k results
  - Query processing
  - Index construction
  - Top-k search
- Social search
Inverted index replication
- Broker forwards query to server with lowest load → high resource costs

Inverted Index partitioning
- By documents
- By terms
  (Work of brokers depends on partitioning strategy)

Variations of LRU strategy for dropping data from cache
Index partitioning strategies

Partitioning by documents ("horizontal partitioning": inverted lists are partitioned)
- Vocabulary is replicated on all servers (i.e., nodes)
- Inverted list entries are hashed onto nodes by document IDs
- Query is forwarded to each node and results are merged
  → easy to maintain, scalable, load-balanced,
  → resource-consuming

Partitioning by terms ("vertical partitioning": vocabulary is partitioned)
- Vocabulary is (partitioned and) distributed across multiple nodes
- Inverted lists are mapped onto nodes responsible for the corresponding terms
- Query is sent to nodes with relevant terms
  What are the consequences for maintenance, scalability, load-balancing, resource-consumption?
Computing top-k results (1)

Top-k join-and-sort for Boolean queries on virtual relations of the form

\[ \text{Index} \ (\text{term}, \text{docID}, \text{Sc}) \]

**Input:** query \( q = t_1 \ t_2 \ ... \ t_l \)

**Required:** top-k docs \( d_1, d_2, ..., d_k \) ranked by some match score:

\[ \forall i, 1 < i \leq k, \forall j > k: \text{Sc}(d_i, q) \leq \text{Sc}(d_{i-1}, q) \land \text{Sc}(d_i, q) \geq \text{Sc}(d_j, q) \]

\[
\text{top-k}\{ \sigma_{[\text{term}=t_1]}(\text{Index}) \bowtie_{\text{docID}} \\
\sigma_{[\text{term}=t_2]}(\text{Index}) \bowtie_{\text{docID}} \\
... \\
\sigma_{[\text{term}=t_l]}(\text{Index}) \text{ order by } \text{Sc} \text{ desc}\}
\]

Most efficient when inverted list entries are sorted by docIDs!
Computing top-k results (2)

**Top-k join with score aggregation** on virtual relations of the form

\[ D_1(\text{docID}, \text{score}_{t_1}), \ldots, D_l(\text{docID}, \text{score}_{t_l}) \]

**Input:** query \( q = t_1 \ t_2 \ \ldots \ t_l \)

**Required:** top-k docs \( d_1, d_2, \ldots, d_k \) ranked by some match score:

\[ \forall i, 1 < i \leq k, \forall j > k: \text{Sc}(d_i, q) \leq \text{Sc}(d_{i-1}, q) \land \text{Sc}(d_i, q) \geq \text{Sc}(d_j, q) \]

Select \( \text{docID}, \ \text{Sc}(D_1.\text{score}_{t_1}, \ldots, D_l.\text{score}_{t_l}) \) As Score

From Outer Join \( D_1, \ldots, D_l \)

Order By Score Limit \( k \)

If \( \text{Sc} \) is monotone, simple and principled algorithms exist.
Top-k processing of score-ordered inverted lists

- **Assumptions**
  - List entries sorted by per-term doc scores
  - Scoring function $Sc(a_1, ..., a_l)$ is monotone
    \[(a_1 \geq b_1) \land \cdots \land (a_l \geq b_l) \Rightarrow Sc(a_1, ..., a_l) \geq Sc(b_1, ..., b_l)\]

- **General heuristics**
  1. Scan lists in sequentially and in Round-Robin fashion (disregard lists with term-idf score below some threshold or prioritize short lists)
  2. If possible (i.e., when the whole lists are in main memory) perform random access to entries with same docID in other lists
  3. Compute scores for docs incrementally, as more dimensions (i.e., per-term scores) are observed
  4. Stop when top-k docs are found (heuristically: until all dimensions are seen for $k' > k$ docs)
Threshold algorithm (Fagin et al. 2001*)

- All inverted lists $L_1, \ldots, L_l$ are sorted by $tf$
- **Random access** to each list is possible

Do **sorted access** in parallel to all lists

Let $\text{cdim}_i$ be the last position visited in **sorted access** in each $L_i$
Define threshold $T = Sc(\text{cdim}_1.score, \ldots, \text{cdim}_l.score)$

If new doc $d$ is seen in one of the lists

- Find all other dimensions of $d$ in all other lists
- Compute overall score $Sc$ of $d$
- If $Sc$ is among top-$k$ highest scores seen so far
  - Store $d$ in top-$k$ buffer (break ties arbitrarily)
- Stop when $k$ docs are found with overall score $Sc > T$

*See: [Optimal aggregation algorithms for middleware](#)
Threshold algorithm (TA): example

Find top-2 results

<table>
<thead>
<tr>
<th>dcoID</th>
<th>Tf1</th>
<th>dcoID</th>
<th>Tf2</th>
</tr>
</thead>
<tbody>
<tr>
<td>79</td>
<td>0.05</td>
<td>53</td>
<td>0.06</td>
</tr>
<tr>
<td>31</td>
<td>0.035</td>
<td>41</td>
<td>0.04</td>
</tr>
<tr>
<td>53</td>
<td>0.03</td>
<td>31</td>
<td>0.028</td>
</tr>
<tr>
<td>41</td>
<td>0.025</td>
<td>11</td>
<td>0.02</td>
</tr>
<tr>
<td>11</td>
<td>0.01</td>
<td>79</td>
<td>0.01</td>
</tr>
</tbody>
</table>

\[ T = 0.11 \]

Top-2 result buffer

<table>
<thead>
<tr>
<th>dcoID</th>
<th>Tf1</th>
<th>dcoID</th>
<th>Tf2</th>
</tr>
</thead>
<tbody>
<tr>
<td>79</td>
<td>0.05</td>
<td>53</td>
<td>0.06</td>
</tr>
<tr>
<td>31</td>
<td>0.035</td>
<td>41</td>
<td>0.04</td>
</tr>
<tr>
<td>53</td>
<td>0.03</td>
<td>31</td>
<td>0.028</td>
</tr>
<tr>
<td>41</td>
<td>0.025</td>
<td>11</td>
<td>0.02</td>
</tr>
<tr>
<td>11</td>
<td>0.01</td>
<td>79</td>
<td>0.01</td>
</tr>
</tbody>
</table>

\[ T = 0.075 \]

Next threshold smaller than any top-k score \(\rightarrow\) stop!

Top-2 result buffer
No Random Access algorithm (Fagin et al. 2001)

- All inverted lists $L_1, \ldots, L_l$ are sorted by $tf$
- No random access

Precompute and maintain $\text{min}_1, \ldots, \text{min}_l$, the smallest possible scores from the lists $L_1, \ldots, L_l$

Do **sorted access** in parallel to all lists

Let $\text{cdim}_i$ be the last position visited in **sorted access** in each $L_i$

Maintain $(\text{cdim}_1.\text{score}, \ldots, \text{cdim}_l.\text{score})$

For every doc $d$ with some unseen dimension

- Compute lower bound $Sc^L$ of $Sc$ by replacing unseen $\text{dim}_i.\text{score}$ by $\text{min}_i$ and upper bound $Sc^U$ of $Sc$ by replacing unseen $\text{dim}_i.\text{score}$ by $\text{cdim}_i.\text{score}$

Maintain top-$k$ docs with highest $Sc^L$ (break ties using $Sc^U$ scores)

Stop when current $Sc^U$ exceeds smallest top-$k$ score
NRA algorithm: example

Find top-2 results

<table>
<thead>
<tr>
<th>dcoID</th>
<th>Tf1</th>
<th>dcoID</th>
<th>Tf2</th>
</tr>
</thead>
<tbody>
<tr>
<td>79</td>
<td>0.04</td>
<td>53</td>
<td>0.06</td>
</tr>
<tr>
<td>31</td>
<td>0.035</td>
<td>41</td>
<td>0.04</td>
</tr>
<tr>
<td>53</td>
<td>0.03</td>
<td>31</td>
<td>0.028</td>
</tr>
<tr>
<td>41</td>
<td>0.03</td>
<td>11</td>
<td>0.02</td>
</tr>
<tr>
<td>11</td>
<td>0.01</td>
<td>79</td>
<td>0.01</td>
</tr>
</tbody>
</table>

53: (0.1 – 0.07)  
79: (0.1 – 0.05)  

Result buffer

<table>
<thead>
<tr>
<th>dcoID</th>
<th>Tf1</th>
<th>dcoID</th>
<th>Tf2</th>
</tr>
</thead>
<tbody>
<tr>
<td>79</td>
<td>0.04</td>
<td>53</td>
<td>0.06</td>
</tr>
<tr>
<td>31</td>
<td>0.035</td>
<td>41</td>
<td>0.04</td>
</tr>
<tr>
<td>53</td>
<td>0.03</td>
<td>31</td>
<td>0.028</td>
</tr>
<tr>
<td>41</td>
<td>0.03</td>
<td>11</td>
<td>0.02</td>
</tr>
<tr>
<td>11</td>
<td>0.01</td>
<td>79</td>
<td>0.01</td>
</tr>
</tbody>
</table>

53: (0.095 – 0.07)  
79: (0.08 – 0.05)  
41: (0.075 – 0.05)  
31: (0.075 – 0.045)  

Result buffer
NRA algorithm: example

Find top-2 results

dcolID | Tf1  
---- | ----  
79   | 0.04 
31   | 0.035 
53   | 0.03 
41   | 0.03 
11   | 0.01 

Result buffer

53: (0.09)  
79: (0.068 – 0.05)  
41: (0.07 – 0.05)  
31: (0.063)  

Result buffer

53: (0.09)  
79: (X)  
41: (X)  
31: (X)  

Result buffer

53: (0.09)  
41: (0.07)  
31: (0.063)  

Result buffer
Instance optimality of TA and NRA

Definition

For class $\mathcal{A}$ of algorithms and class $\mathcal{D}$ of datasets, algorithm $B \in \mathcal{A}$ is instance optimal over $(\mathcal{A}, \mathcal{D})$ if for every $A \in \mathcal{A}$ and every $D \in \mathcal{D}$:

$$\text{cost}(B, D) \leq c \times \text{cost}(A, D) + c' \iff \text{cost}(B, D) = O(\text{cost}(A, D))$$

It can be shown:

- For any monotone scoring function, TA and NRA correctly retrieve the top-k results.
- TA is instance optimal over all algorithms that are based on sorted and random accesses to inverted lists (no „wild guesses“).
- NRA is instance optimal over all algorithms with sequential accesses only.
Implementation issues

- **Priority queues**
  - Empirically, bounded-size priority queues show better performance than Fibonacci heaps

- **Memory management**
  - Memory load is very important for efficiency (similarly to scan depth)
  - Early candidate pruning is important

- **Hybrid block index**
  - Group inverted list entries in blocks and sort blocks by scores
  - Keep entries within a block in docID order
  - After each block read: merge-join first, then update priority queue
“Champion lists” heuristics (Brin & Page 1998)

- All inverted lists $L_1, \ldots, L_l$ are sorted by doc authority (e.g., PageRank) scores
- Keep additional lists $F_1, \ldots, F_l$ (champion lists) with docs having $tf$ scores above some threshold in each dimension

Compute scores for all docs in $\cap_i F_i$ and keep top-$k$ results;

$$Cand := (\bigcup_i F_i) \setminus (\cap_i F_i)$$

For each $d \in Cand$ do

- Compute partial score of $d$

Scan inverted lists $L_i$ in Round-Robin fashion

- If $dim_i.doc \in Cand$
  - Add $dim_i.score$ to partial score of $dim_i.doc$
- Else
  - Add $dim_i.doc$ to $Cand$ and set its partial score to $dim_i.score$

Terminate when $k' > k$ docs with complete scores are found;
Probabilistic approximate top-k processing

- Makes use of
  - certain score distribution in each of the inverted lists (approximated by histograms)
  - pair-wise convolution of score distributions

\[
\sum_{0 \leq i \leq d} B_{t_1}[i].freq \ast B_{t_2}[d - i].freq = B_{t_1+t_2}[d].freq
\]

- correlation between scores in different dimensions
- probabilistic inequalities for stopping conditions
Feature overview of top-k algorithms

Summary

- Distributed index maintenance
  - Horizontal partitioning (by documents)
    - High costs, easy to maintain, scalable, load-balanced
  - Vertical partitioning (by terms)
    - Low costs, maintenance and load-balancing are difficult

- Top-k algorithms
  - Join and sort when list entries are sorted by docIDs
  - When list entries sorted by per-term doc scores:
    - Top-k join with score aggregation
    - “Champion lists” (uses lists with authority scores)
    - Threshold algorithm
    - No Random Access algorithm

- Probabilistic approximate top-k processing
  - Estimation of unseen scores by convolution of score distributions in inverted lists