

# Exploiting Correlated Keywords to Improve Approximation Filtering

---

Christian Zimmer, Christos  
Tryfonopoulos and Gerhard Weikum

In SIGIR 2008



# Structure

---

- Overview
- Main problem
- Example
- Hash Sketches
- KMV Synopses
- System architecture
- Correlation
- The algorithm USS
- The algorithm CSS
- Multi-key statistics
- Experimental evaluation
- Conclusions



# Overview

---

- In an **Information Filtering** scenario users express their interests via subscriptions and get notified when events matching their subscriptions are published
- **Exact Information Filtering (IF)** scenarios involve delivering notifications from **every** publisher to subscribers
- In **Approximate IF** only a few publishers store the user query and are monitored for new events



# Main problem

---

- Exact IF model imposes an information overload burden on the user
- **Main issue:** The careful selection of few promising publishers to store user query
- Subscribers use statistical metadata to identify promising publishers

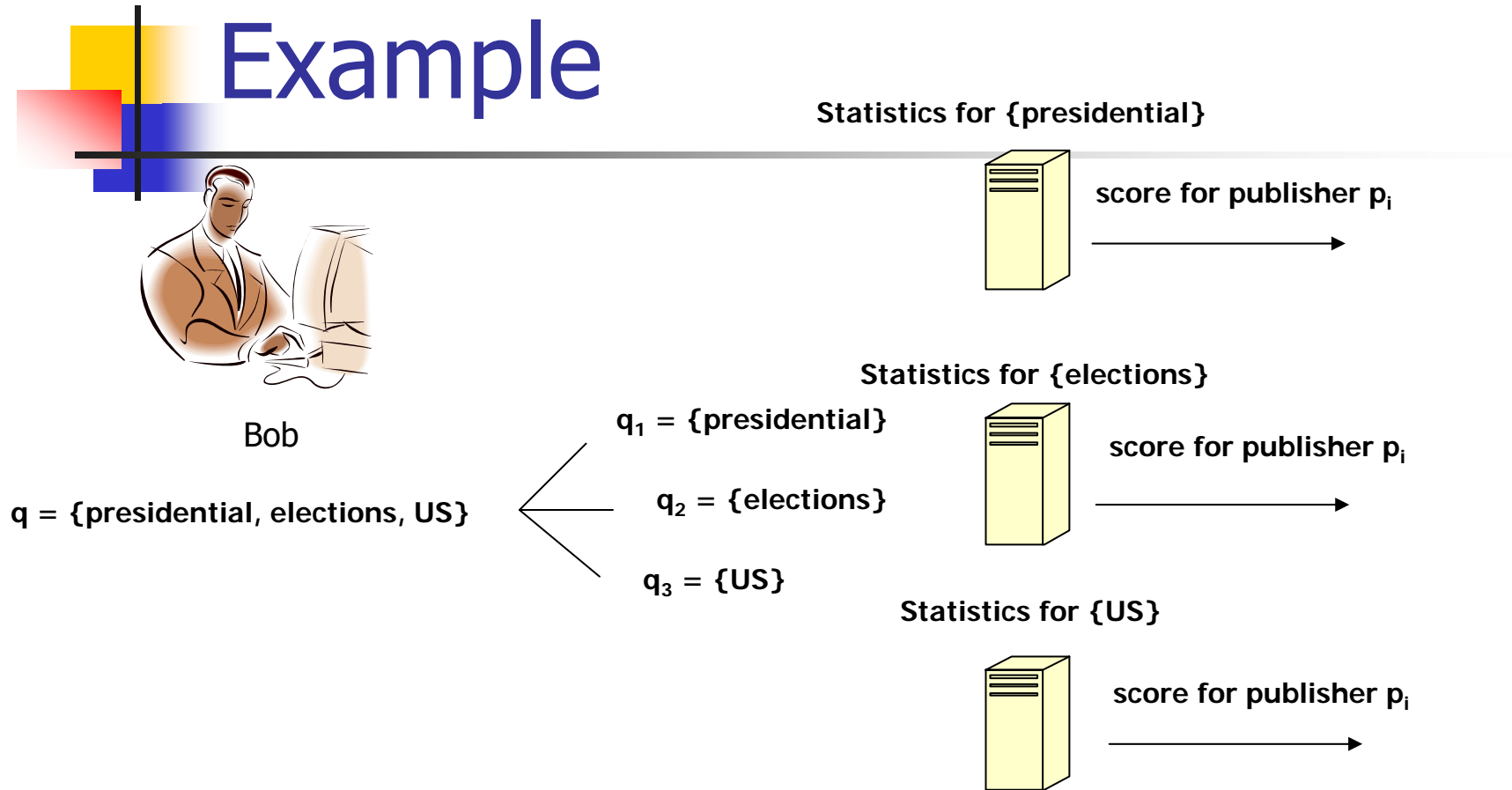


# Main problem (continue)

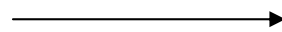
---

- Statistics are maintained in a directory on a **per-keyword** basis
- Possible correlation between keywords is disregarded
- This work:  
Improves approximate IF in a distributed setting by exploiting correlation between keywords

# Example



aggregating individual scores



ranking publishers

# Example (continue)



Bob

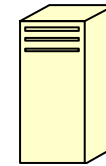
$q = \{\text{presidential, elections, US}\}$

$q_1 = \{\text{presidential, elections}\}$

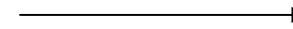
$q_2 = \{\text{elections, US}\}$

$q_3 = \{\text{US}\}$

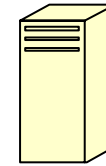
Statistics for {presidential, elections}



score for publisher  $p_i$



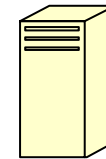
Statistics for {elections, US}



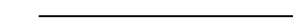
score for publisher  $p_i$



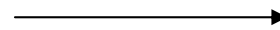
Statistics for {US}



score for publisher  $p_i$



**aggregating individual scores +  
prediction scores**



**ranking publishers**



# Structure

---

- Overview
- Main problem
- Example
- Hash Sketches
- KMV Synopses
- System architecture
- Correlation
- The algorithm USS
- The algorithm CSS
- Multi-key statistics
- Experimental evaluation
- Conclusions





# Hash Sketches

---

- Usage: probabilistic **estimation** of the cardinality of a multi-set  $S$  (distinct value estimation).
- A number of input values are spread over a number of output values via a hashing function.
- Hash Sketch of the union of multi-sets  $A$  &  $B$  is the bit-wise OR between Hash Sketch of  $A$  and Hash Sketch of  $B$ .
- To compute the intersection of  $A$  &  $B$ , we use:  
$$|A \cap B| = |A| + |B| - |A \cup B|$$
- **But:** Hash Sketches for multi-key queries impose inaccuracy



# KMV Synopses

---

- Assume  $D$  points are placed uniformly on the unit interval
  - The expected distance between two neighboring points is  $1/(D+1) \approx 1/D$
  - The expected value of  $U_k$  ( $k$ -th smallest point) is  $E[U_k] \approx k/D$
  - Thus, a basic DV estimator for  $D$  is  $D = k/U_k$
- Let  $S$  be a multi-set and  $\theta(S)$  the domain of its distinct values.
- By applying a hash function  $h()$  to each value of  $\theta(S)$   $h: \theta \rightarrow \{0, 1, \dots, M\}$ , the  $k$  smallest hashed values are recorded. (KMV Synopsis for  $k$  minimum values)



# KMV Synopses (continue)

---

- Let  $A, B$  be multi-sets,  $L_A, L_B$  their KMV synopses of size  $k_A, k_B$  respectively and  $L$  the KMV synopses of their intersection
- DV estimator for **union** is  $D_{\cup} = (k-1)/U_k$ 
  - where  $k = \min(k_A, k_B)$
- DV estimator for **intersection** is  $D_{\cap} = (K_{\cap}/k) \cdot (k-1)/U_k$ 
  - where
    - $k = \min(k_A, k_B)$
    - $K_{\cap} = |\{u \in V_L : u \in \theta(A) \cap \theta(B)\}|$
    - $L = \{h(v_1), h(v_2), \dots, h(v_k)\}$
    - $V_L = \{v_1, v_2, \dots, v_k\}$



# System architecture

---

- System consists of three components:
  - Publishers
  - Directory nodes
  - Subscribers
- Distributed directory maintained by super-peers



# Publishers

---

- A publisher wants to expose its content to the system in the form of **per-key statistics (posts)**
- Statistics consist of inverted lists of documents (maintained as Hash Sketches or KMV Synopses)
- A publisher sends its statistics to directory nodes **periodically** to help the ranking procedure made by subscribers
- Also, it is responsible for maintaining subscribers' continuous queries



# Directory nodes

---

- Directory nodes store statistics about the publishers' local contents
- They make them available to the subscribers
- Each node is responsible for a particular subset of keys existing in IF system
- Key data set is partitioned using a DHT hash function
- Directory nodes are organized using a **Chord DHT** forming



# Subscribers

---

- A subscriber seeks for publishers that will publish interesting documents **in the future**
- In order to subscribe to a publisher  $p$ , subscriber forwards its continuous query  $q$  to  $p$
- There  $q$  is matched with every publication of  $p$



# Subscribers (continue)

---

- Let  $q = \{k_1, k_2, k_3, \dots, k_n\}$  be a continuous query made by a subscriber  $s$
- Subscriber  $s$ 
  - contacts directory nodes to retrieve statistics for every key  $k_j$
  - ranks publishers and sends  $q$  only to top-ranked ones
- **Only** publishers that store  $q$  will match their content against  $q$
- But: considering individual key statistics and not key set statistics leads to **reduced recall**





# Structure

---

- Overview
- Main problem
- Example
- Hash Sketches
- KMV Synopses
- System architecture
- Correlation
- The algorithm USS
- The algorithm CSS
- Multi-key statistics
- Experimental evaluation
- Conclusions



# Correlation

---

- The probability that a random document contains key  $a$  given that it contains key  $b$  is:

$$P(A|B) = \frac{df(ab)/|D|}{df(b)/|D|} = \frac{df(ab)}{df(b)}$$

- Let  $S = \{k_0, k_1, k_2, \dots, k_n\}$  be a correlated key set. The probability that a random document contains  $k_0$  given that it contains all other keys is:

$$P(K_0|K_1 \dots K_{n-1}) = \frac{df(k_0 k_1 \dots k_{n-1})}{df(k_1 \dots k_{n-1})}$$



# Correlation (continue)

---

- All continuous queries can be considered candidates for harvesting **multi-key** statistics
- Consider a key pair  $ab$  that has **no** correlation
  - We consider it as interesting, if  $P(A|B)$  and  $P(B|A)$  are below some threshold  $\beta$
- Interesting keyword sets
  - with **uncorrelated** keywords
  - with **anti-correlated** keywords



# The algorithm USS

---

- Let  $s$  be a subscriber that subscribes a continuous query  $q = \{k_1, k_2, \dots, k_n\}$  in the IF system
- The following steps are executed
  1. For each key  $k_j$ ,  $1 \leq j \leq n$ , subscriber  $s$  contacts directory node  $d(k_j)$  and retrieves the statistical information for key  $k_j$
  2. For publisher  $p_i$  appearing in all statistics,  $s$  computes an estimation of  $df_i(q)$  using **synopses intersection techniques** and applies **prediction techniques** to compute a behavior prediction score



# The algorithm USS (continue)

---

3. Subscriber  $s$  sends the query  $q$  only to top-ranked publishers. These publishers **only** will store  $q$
4. Due to dynamics in publishing, steps 1-3 repeat in a **periodic** way



# The algorithm CSS

---

- USS approach has problems because of **single-key** statistics
  - higher network load  
The directory has to send long lists to the subscriber
  - inaccuracy  
synopsis for the intersection of documents containing all keys
  - prediction errors  
single-key statistics introduce additional errors
- The algorithm CSS introduces the idea of maintaining **multi-key** statistics



# The algorithm CSS (continue)

---

- Let  $S = \{k_1, k_2, \dots, k_n\}$  be a key set. We employ a deterministic function to select a directory node  $d(S)$  and be responsible for this set:
  1.  $d(S)$  contacts all directory nodes responsible for key  $k_j \in S$  and retrieves the synopses from all documents containing that key
  2.  $d(S)$  computes intersections among synopses and computes  $df(S)$
  3.  $d(S)$  then computes the conditional probabilities for each key  $k_j$



# Multi-key statistics

---

- Multi-key statistics can be piggybacked on messages that need to be send anyway
- Example:
  - Assume that  $d(S)$  responsible for key  $k$  identified key set  $S$  as useful
  - Whenever a publisher  $p$  updates its statistics for key  $k$ ,  $d(S)$  can inform  $p$  about key set  $S$





# Multi-key statistics (continue)

---

- Idea: computing statistics for a multi-key query by combining statistics from **subsets** available in the directory

- Scoring function for calculating publisher's score:

$$\text{score}_s(p) = \sum_{S_i \subseteq S} |S_i| \cdot \text{predScore}_{S_i}(p)$$

where  $\text{predScore}_{S_j}(p)$  represents the likelihood that  $p$  will produce a document containing  $S_j$  **in the future**

- Intuition behind weighting prediction score with  $|S_j|$ : prediction score for small subsets dominates the sum



# Structure

---

- Overview
- Main problem
- Example
- Hash Sketches
- KMV Synopses
- System architecture
- Correlation
- The algorithm USS
- The algorithm CSS
- Multi-key statistics
- Experimental evaluation
- Conclusions

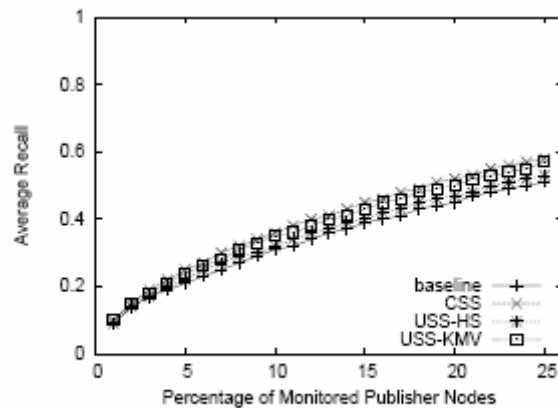


# Experimental evaluation

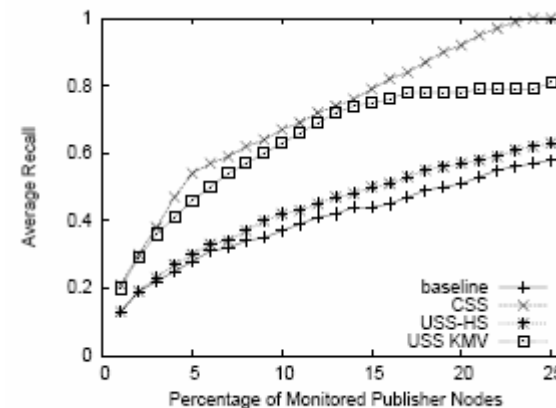
---

- Recently proposed benchmark for evaluating p2p information retrieval
  - Data set: 800,000 web documents from Wikipedia
  - Algorithm: distributing documents among 1,000 publishers with **controlled overlap**
- Continuous queries with one, two and three keys
- Queries are indexed in up to 25% of publishers
- Use of **publication rounds**; publishing 400 documents per round
- Evaluation metric:  
**average recall** = total number of notification received/total number of documents matching the subscriptions

# Results (1)



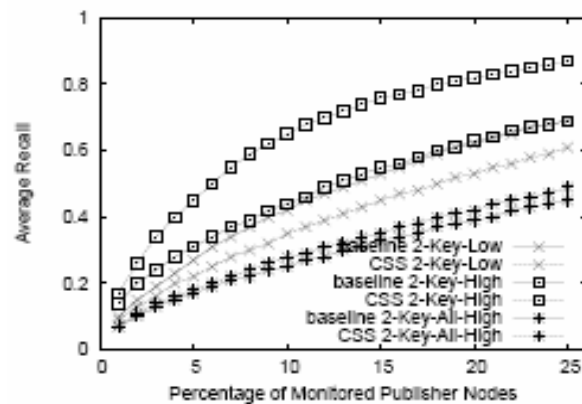
(a) Two-key queries



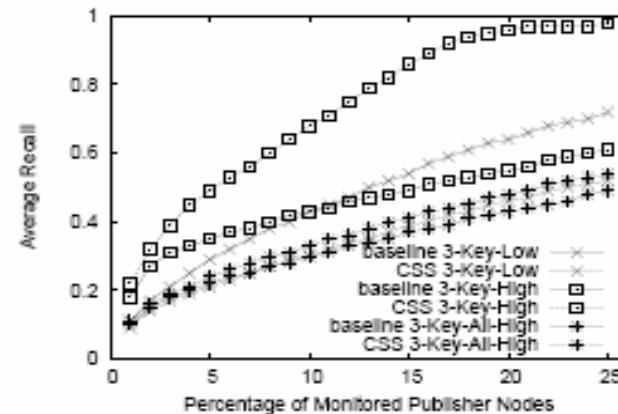
(b) Three-key queries

- For two-key queries:
  - baseline algorithm: monitoring 24% of publishers to achieve average recall = 0.5
  - CSS algorithm: monitoring 19% of publishers to achieve equal levels of recall
- Three-key queries are more selective with less matching documents, achieving higher improvements of recall
- USS-KMV algorithm outperforms USS-HS algorithm. USS-HS suffers from inaccuracy of combining Hash Sketches of more than 2 sets.

## Results (2)



(e) Effect for two-key queries



(f) Effect for three-key queries

- CSS algorithm has no significant effect for key sets where all keys are highly correlated. But it significantly improves filtering for key sets with low correlations
- Key sets where at least one key is highly correlated to all others show great gains of improvement



# Conclusions

---

- The CSS algorithm that exploits multi-key statistics outperforms competitors achieving high average recall scores
- The usage of KMV synopses instead of Hash Sketches improves effectiveness of USS algorithm
- Multi-key queries with uncorrelated keys achieve high gains of recall



# Conclusions (critic)

---

- Approximate IF achieves high **scalability**; faster response time and lower message traffic
  - But: we have a **loss** in recall
- Exploiting **correlation** between keywords achieves high **gains** of recall (esp. for uncorrelated keywords)
  - But: multi-key queries lead to dimensional curse
- Multi-key statistics allow subscribers to select promising publishers **easier**
  - But: what about **correlated** databases
  - **Difficulty** in choosing interesting key sets



# Thank you for your attention

---

## Questions?







# My question

---

Multi-key queries with correlated keys show low improvements of recall when CSS algorithm is used. Why?