Exploiting Correlated Keywords to Improve Approximate Information Filtering

Christian Zimmer, Christos Tryfonopoulos, and Gerhard Weikum
Databases and Information Systems Department
Max-Planck Institut for Informatics
66123 Saarbrücken, Germany
{czimmer, trifon, weikum}@mpi-inf.mpg.de

ABSTRACT

Information filtering, also referred to as publish/subscribe, complements one-time searching since users are able to subscribe to information sources and be notified whenever new documents of interest are published. In approximate information filtering only selected information sources, that are likely to publish documents relevant to the user interests in the future, are monitored. To achieve this functionality, a subscriber exploits statistical metadata to identify promising publishers and index its continuous query only in those publishers. The statistics are maintained in a directory, usually on a per-keyword basis, thus disregarding possible correlations among keywords. Using this coarse information, poor publisher selection may lead to poor filtering performance and thus loss of interesting documents.[1]

Based on the above observation, this work extends query routing techniques from the domain of distributed information retrieval in peer-to-peer (P2P) networks, and provides new algorithms for exploiting the correlation among keywords in a filtering setting. We develop and evaluate two algorithms based on single-key and multi-key statistics and utilize two different synopses (Hash Sketches and KMV synopses) to compactly represent publishers. Our experimental evaluation using two real-life corpora with web and blog data demonstrates the filtering effectiveness of both approaches and highlights the different tradeoffs.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed applications*; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Selection process, Information filtering*

General Terms

Algorithms, Design, Performance

Keywords

Peer-to-Peer (P2P), information systems, approximate publish/subscribe, distributed information filtering (IF), distinct-value (DV) estimation

[1]This work has been partly supported by the EU project AEOLUS.

SIGIR'08, July 20–24, 2008, Singapore.

1. INTRODUCTION

In an information filtering (IF) scenario a subscriber submits a continuous query (or *subscription*) and waits to be notified from the system about certain events of interest that take place (i.e., about newly published documents relevant to the continuous query). Most approaches to IF taken so far have the underlying hypothesis of potentially delivering notifications from *every* information producer to subscribers [26, 25, 2]. This exact publish/subscribe model imposes an information overload burden on the user, and also creates an efficiency and scalability bottleneck that is probably not desirable in applications like news or blog filtering.

In *approximate* information filtering only a few carefully selected, specialized, and promising publishers store the user query and are monitored for new publications. Thus, the user query is replicated to these sources and only published documents from these sources are forwarded to the subscriber. The system is responsible for managing the user query, discovering new potential sources and moving queries to better or more promising sources. Since in an IF scenario the data is originally highly distributed residing on millions of sites (e.g., with people contributing to blogs), approximate IF seems an ideal candidate for such a setting. This is also supported by the fact that exact IF functionality has proven expensive for such distributed environments [26, 25, 2]. Thus, approximate IF achieves much better scalability of such systems by trading faster response times and lower message traffic for a moderate loss in recall.

In approximate IF the publisher selection for a given continuous query with multiple keywords is driven by statistical summaries that are stored by the system. These summaries are provided to the directory by the publishers and can be managed in different ways ranging from centralized solutions like servers or server farms, to super-peer or pure peer-to-peer (P2P) solutions in the form of a distributed P2P directory built on top of a DHT [1, 24] or other kinds of overlay networks. For scalability, the summaries have publisher granularity, not document granularity, thus capturing the best publisher for certain keywords (also referred to as *keys*) but not for specific documents. This, together with per-key organization of the directory that disregards keyword correlations (also referred to as *key sets*) are two of the basic reasons that may possibly lead to insufficient recall. On the other hand, considering statistics for *all* possible key sets is clearly not possible due to the explosion in the feature space.

As an example scenario, consider a user Bob who wants to follow the discussion about the *presidential elections* in the *US*, and wants to receive notifications from a number of different sites like news agencies, portals, and user blogs. Clearly, Bob would be interested in monitoring a variety of publishers but is not interested in receiving all the articles published by all sources, as it would be the case for exact IF. Thus, in an approximate IF scenario, Bob would sub-

mit the continuous query *US presidential elections* to the filtering system. The basic approach would decompose the continuous query into the three individual keys and use the statistics from the directory to compute a combined score (e.g., intersection or some other kind of aggregation of individual key scores) for each key and publisher. This score would represent the probability of each source to publish documents about US presidential elections in the near future. This approach may lead to poor filtering quality as the top-ranked publishers for the *complete* query may not be among the top selected publishers. In the worst case, a selected publisher may deliver many documents for each single keyword, but no single document matching *all keywords*, since this information is not present in the directory.

In this paper, we develop and evaluate two approaches that use correlations among keys to improve filtering quality in the scenario described above. The first algorithm (coined USS) uses existing single-key synopses stored in the directory to estimate the publishing behavior of information sources for key sets, while the second (coined CSS) enhances the directory to explicitly maintain statistical metadata about selectively chosen key sets.

Our work builds upon previous work in the area of information retrieval (IR) over P2P networks [18] and extends it to the approximate IF setting in various novel ways. Our contribution can be summarized as follows:

- In contrast to distributed IR settings for one-time searching where sources are ranked according to their document collections, in approximate IF the publishers are ranked according to their probability to publish relevant documents in the near future, which poses different requirements for the maintenance of statistics. This is the first paper to develop algorithms for exploiting keyword correlations in such an IF setting.
- We extend the approach of [18] for two-key queries, to the case of multi-key continuous queries for an arbitrary number of keys. We also provide new algorithms to approximate multi-key statistics by combining the statistics of arbitrary subsets.
- We show that Hash Sketches, used in [18] for compactly representing the documents, yield inaccurate results when considering continuous queries with more than two keys. We, thus, propose the usage of very recent state-of-the-art techniques for compact representation of multisets [6]. These new techniques allow us to compute accurate synopses for multi-key queries, and improve the filtering effectiveness.

The remainder of this paper is organized as follows. Section 2 introduces background information on Hash Sketches, and KMV synopses. Section 3 discusses the architecture of our approximate IF system. In Section 4 we outline measures of correlation for keywords and present the two proposed algorithms. Section 5 shows our experimental results. Section 6 discusses related research. Finally, Section 7 concludes the paper.

2. BACKGROUND

Synopses for compact approximation of sets, multisets, and their statistical properties have recently received great attention in the context of sensor networks, data streams, content delivery, and estimation issues in structured databases [10]. The methods under consideration include Bloom filters [7], Hash Sketches [13], min-wise independent permutations [8], and KMV synopses [6], all of which are hash-based but differ in their strengths and limitations of representing various kinds of properties. The latter include set membership testing, set cardinality estimation, estimating the number of distinct elements in a multiset, forming intersections, unions, differences, etc. As shown in [18], these compact representations are a necessary component also for distributed IR approaches, since they assist in lowering the volume of data transferred over the network and thus increase system scalability. Following this line, we also utilize these techniques for compactly disseminating statistics about the publishers. In this section we give brief background information on Hash Sketches and KMV synopses that are utilized by our algorithms.

2.1 Hash Sketches

Hash Sketches denote a well-known statistical tool for probabilistically estimating the cardinality of a multiset S. This distinct-value estimation technique was proposed in [13] and further improved in [11]. Hash Sketches rely on the existence of a pseudo-uniform hash function which spreads input values pseudo-uniformly over its output values.

A key property of Hash Sketches lies in the ability to combine them. We can derive the Hash Sketch of the **union** of an arbitrary number of multisets from the Hash Sketches of each multiset by taking their bit-wise OR. Thus, given the compact synopses of a set of multisets, one can instantly estimate the number of distinct items in the union of these multisets.

Furthermore, Hash Sketches can be used to estimate the cardinality of the **intersection** (overlap) of two sets. First, recall that $|A \cap B| = |A| + |B| - |A \cup B|$. Second, by utilizing the union method outlined above, one can derive the Hash Sketch for $A \cup B$, and thus compute the cardinality of $|A \cap B|$. However, it is not possible to create the Hash Sketch synopsis of the intersection for future use. The above can be generalized to more than two sets, using the inclusion-exclusion principle and the sieve formula by Poincaré and Sylvester. Obviously, to compute the intersection of a huge number of Hash Sketches, the relative error is propagated and the distinct-value estimation is getting inaccurate. Even regarding the overlap of three multisets, the sieve formula needs a high computation complexity.

2.2 KMV Synopses

In [6], the KMV synopses and appropriate DV estimators are introduced. The main difference to Hash Sketches are constituted in the lower computational costs and the more accurate DV estimation. Especially, the main focus of KMVs is on arbitrary multiset operations including union, intersection, and differences. First, we explain the main motivation behind the DV estimators. Then, we introduce the KMV data structure and the basic DV estimator. Using the KMV synopses and the basic estimator, the multiset operations union and intersection can be applied.

Assume that D points are placed randomly and uniformly on the unit interval. The expected distance between two neighboring points is $1/(D+1) \approx 1/D$, such that the expected value of U_k, the k-th smallest point, is $E[U_k] \approx k/D$. Thus $D \approx k/E[U_k]$. If we know U_k itself, a basic estimator for the number of points as proposed in [4] is

$$\hat{D}_k^b = k/U_k \tag{1}$$

In the DV estimation problem, we have an enumeration of distinct values $v_1, v_2, \ldots, v_D$ in dataset A with domain $\theta(A)$. Using a hash function $h : \theta(A) \mapsto \{0, 1, \ldots, M\}$ such that the sequence $h(v_1), h(v_n), \ldots, h(v_D)$ looks like a sequence of independent and identically distributed samples from the discrete uniform distribution on $\{0, 1, \ldots, M\}$. Assuming that M sufficiently greater than D, the sequence $U_1 = h(v_1)/M, U_2 = h(v_2)/M, \ldots, U_D = h(v_D)/M$ will approximate the realization of a sequence of samples from the continuous uniform distribution on $[0, 1]$. The requirement that M is much larger than D avoids collisions and ensures with high probability, $h(v_i) \neq h(v_j)$ for all $i \neq j$.

2.2.1 Creating KMV Synopsis and DV Estimator

Using the idea of the basic estimator introduced in [4], a KMV synopsis for a multiset S is created as described in [6]: by applying the hash function $h()$ to each value of $\theta(S)$, the k smallest of the hashed values are recorded. This *simple* synopsis (e.g., set L_S of hashed values) is called KMV synopsis (for k *minimum values*).

In [6], the DV estimator for KMV synopses extends the basic estimator 1 and uses the following computation:

$$\hat{D}_k = (k-1)/U_k \tag{2}$$

It is shown that, under the assumption that $D > k$, this estimator is unbiased (in contrast to Equation 1) and $\hat{D}_k$ is used for the multiset operations described below. But, if $D \leq k$, then it is easily possible to detect this situation and return the exact value of D from the synopsis.

2.2.2 Multiset Operations on KMV Synopsis

So far, we have an estimator for KMV synopses. Now, we focus on multiset operations using two or more KMV synopses in combination with estimating the compound set and present the union and intersection operation.

Throughout, all synopses are created using the same hash function $h : \theta \mapsto \{0, 1, \ldots, M\}$ where θ denotes the data value domain appearing in the synopses and $M = O(|\theta|^2)$. Ordinary set operations are denoted by $\{\cup, \cap\}$ and multiset operations by $\{\cup_m, \cap_m\}$.

Union Operation: We assume two multisets A and B with their KMV synopses L_A and L_B of size k_A and k_B, respectively. The goal is to estimate the number of distinct values in the union of A and B as $D_\cup = |\theta(A \cup_m B|)$. Here, $\theta(S)$ denotes the set of DVs in multiset S. Thus, $D_\cup$ can also be interpreted as $D_\cup = |\theta(A) \cup \theta(B)|$.

Let $L = L_A \oplus L_B$ be defined as the set including the k smallest values in $L_A \cup L_B$, where $k = min(k_A, k_B)$ and L is the KMV synopsis of size k describing $L_A \cup_m L_B$. Thus, by applying the DV estimator for KMV synopses, $D_\cup$ is estimated by following equation:

$$\hat{D}_\cup = (k-1)/U_k \tag{3}$$

Using the symmetric and associative operator $\oplus$, this result can be extended to multiple sets: $L = L_{A_1} \oplus L_{A_2} \oplus \cdots \oplus L_{A_n}$ estimates the number of DVs in $A_1 \cup_m A_2 \cup_m \cdots \cup_m A_n$.

Intersection Operation: As before, consider two multisets A and B with corresponding KMV synopses L_A and L_B of sizes k_A and k_B, respectively. The goal is to estimate $D_\cap = |\theta(A \cap_m B)| = |\theta(A) \cap \theta(B)|$. Set $L = L_A \oplus L_B$ with $L = \{h(v_1), h(v_2), \ldots, h(v_k)\}$, where $k = min(k_A, k_B)$. Each value v_i is an element of $\theta(A) \cup \theta(B)$. Also set $V_L = \{v_1, v_2, \ldots, v_k\}$ and $K_\cap$ as follows:

$$K_\cap = |\{v \in V_L : v \in \theta(A) \cap \theta(B)\}| \tag{4}$$

Obviously, $v \in \theta(A) \cap \theta(B)$ if and only if $h(v) \in L_A \cap L_B$ such that $K_\cap$ can be computed from L_A and L_B alone. $K_\cap$ is utilized to estimate $D_\cap$ using the Jaccard Distance $\rho = D_\cap / D_\cup$ estimated by $\hat{\rho} = K_\cap / k$, the fraction of sampling elements in $V_L \subseteq \theta(A \cup B)$ that belong to $\theta(A \cap B)$. This leads to the proposed estimator:

$$\hat{D}_\cap = (K_\cap / k) \cdot (k-1)/U_k \tag{5}$$

3. SYSTEM ARCHITECTURE

Our approximate IF architecture consists of three components: the *directory*, the *publishers* and the *subscribers*. The notation used in the following sections is summarized in Table 1.

Symbol	Explanation
$\|X\|$	Number of distinct documents in a multiset X
D	Set of documents in the system
D_i	Set of documents on publisher p_i
a, b	Individual keys
ab	Key set (both of a and b)
$D(a)$	Set of documents in D containing key a
$D_i(a)$	Set of documents in D_i containing key a
$df(a)$	Frequency of key a in D $(= \|D(a)\|)$
$df_i(a)$	Frequency of key a in D_i $(= \|D_i(a)\|)$
$SYN(a)$	Synopsis representing documents in $D(a)$
$SYN_i(a)$	Synopsis representing documents in $D_i(a)$
$d(a)$	Directory node responsible for key a

Table 1: Summary of Notation

Publishers. Publishers are information sources that want to expose their content to the IF system, and can be user blogs, digital libraries, or peers with local crawlers that perform focused crawling at portals of their interest. Each publisher exposes its content to the system in the form of per-key statistics (called *posts*) about its local index. These statistics consist of inverted lists of documents relevant to the corresponding key, and are made available to the directory. The posts contain contact information about publishers, together with statistics to calculate quality measures for a given key (e.g., frequencies). Typically, such statistics include also the length of the inverted list and other quality measures to support the publisher ranking procedure carried out by subscribers. Specifically, the size of the inverted list for a key – that is, the frequency for key a, or $df_i(a)$ – can be maintained in the form of a Hash Sketch or KMV synopsis; a publisher inserts an identifer for each document contained in its collection into a local Hash Sketch or KMV synopsis for this key to obtain $SYN_i(a)$. The publishers have to update the directory statistics after a certain number of publications by sending the new summaries to the directory. Finally, publishers are responsible for locally storing continuous queries submitted by subscribers and matching them against new documents they publish. More than one publisher may be used to expose the contents of large digital libraries, portals or blog hosting sites.

Directory. The main functionality of the directory is to store compact, aggregated metadata about the publishers' local indexes, and make these statistics available when requested by the subscribers. This information will be used by the subscribers to determine which publishers are promising candidates to satisfy a given continuous query in the future. There are different ways to implement this type of directory, ranging from centralized solutions that emphasize accuracy in statistics and rely on server farms, to two-tier architectures, as in super-peer systems, that emphasize scalability and fault-tolerance. In our approach, we utilize a distributed directory maintained by super-peers that are organized using a Chord DHT [24] forming a conceptually global, but physically distributed directory that manages the statistics provided by the publishers in a scalable manner with good properties regarding system dynamics (e.g., churn). The DHT is used to partition the key space, such that every super-peer is responsible for the statistics of a randomized subset of keys within the directory. Since there is a well-defined directory node responsible for each key (through the DHT hash function), the synopses representing the index lists of all publishers for a particular key a are *all* sent to the same directory node $d(a)$. Thus, $d(a)$ can compute a moving-window estimate for the global df value of a – $df(a)$ – by performing an union operation for all synopses $SYN_i(a)$ sent by every publisher p_i for key a.

Subscribers. Subscribers are information consumers that seek to satisfy their long-term information needs by subscribing to publishers that will publish interesting documents in the future. To subscribe to a potentially promising publisher p, a subscriber forwards to p the related continuous query, which is stored at the publisher side, and is matched with every new publication that takes place. Sub-

scribers utilize directory statistics to score and rank publishers, and this scoring is based on appropriate publisher selection and behavior prediction strategies [29] that utilize time-series analysis of statistics. To follow the changes in the publishing behavior of information producers, subscribers periodically re-rank publishers by obtaining fresh statistics from the directory, and use the new publisher ranking to reposition their continuous queries.

3.1 Subscription Protocol

Let us assume that a subscriber s wants to subscribe with a continuous query containing multiple keywords. To do so, s needs to determine which publishers in the network are promising candidates to satisfy the continuous query $q = \{k_1, k_2, \ldots, k_n\}$ with appropriate documents published in the future.

To collect statistics about the publishers, s needs to contact the directory to retrieve statistics for all keys $k_i, 1 \leq i \leq n$. Based on the collected statistics, a ranking of publishers is determined and the highest ranked publishers are candidates for storing q. Thus, only publications occurring *in those publishers* will be matched against q and create appropriate notifications. Publishers that make available documents relevant to q, but not indexing q, will not produce any notification, since they are not aware of q. When the publishers that will store q have been determined, s sends the continuous query to them. A publisher p receiving q, will store q in its local database using appropriate query indexing mechanisms such as [28].

This *baseline approach* intersects the statistics for the single keys, i.e., sends the continuous query only to (a subset of) the publishers that published (or will publish) statistics for *all* queried keys. However, this approach may lead to reduced recall, since a publisher appearing in publisher lists for both keys k_i and k_j will not necessarily publish documents containing both k_i and k_j. Thus, to select an appropriate publisher for q, we have to consider the statistics of the key set to determine more accurately its future publishing behavior. Obviously, the larger the subset of q we maintain statistics for, the more accurate our prediction about the behavior of the publisher will be.

Finally, notice that filtering and publisher selection are dynamic processes, and therefore periodic query repositioning, based on user-set preferences, is necessary to adapt to changes in publisher's behavior. To reposition an already indexed query q, a subscriber would re-execute the subscription protocol, acquire new publisher statistics, compute a new ranking, and appropriately modify the set of publishers that will index q.

3.2 Publication and Notification Protocol

When a document d is published by p, it is matched against p's local query database using an appropriate algorithm [28] to determine which subscribers should be notified. Then, for each subscriber s, p sends a notification to s using the contact address associated with the stored query.

4. ALGORITHMS FOR CORRELATION

In this section we summarize the existing measures for correlated key pairs, extend these measures for capturing relatedness among keys in key sets, and explain the correlation model that will drive the extended synopses construction and publisher selection. Subsequently, we present two new algorithms, coined USS and CSS, for exploiting the correlations among key sets in IF scenarios.

4.1 Correlation measures

In [18], the conditional probability that a random document contains a key a given that it contains a key b was introduced as an asymmetric measure of relatedness. Using this measurement, there is no need for knowing or estimating the total number of documents, since the estimator for key sets with two keys a and b is given by:

$$\hat{P}(A|B) = \frac{df(ab)/|D|}{df(b)/|D|} = \frac{df(ab)}{df(b)} \quad (6)$$

Subsequently, [18] proposes methods to identify sufficiently frequent and interesting key sets, and select those that present the lowest correlation. The output of this process would then be used to guide query routing to appropriate nodes.

Here, we modify the approach summarized above to consider arbitrary numbers of keys and propose an approach to identify appropriate key sets in an information filtering scenario. Thus, to consider an arbitrary number of keys in a correlated key set $S = \{k_0, k_1, \ldots, k_{n-1}\}$ and to compute the probability estimator that a random document contains k_0 given that it contains all other keys, the previous formula has to be modified as follows:

$$\hat{P}(K_0|K_1 \ldots K_{n-1}) = \frac{df(k_0 k_1 \ldots k_{n-1})}{df(k_1 \ldots k_{n-1})} \quad (7)$$

Notice that, in an information filtering setting, the continuous queries are actually long-standing queries that will be assessed several times in the future. Thus, all continuous queries can be effectively considered as candidate key sets for harvesting multi-key statistics. Should a more selective system behavior be intended (e.g., for performance reasons), the submitted continuous queries can be further analyzed using frequent itemset mining techniques [3, 12] to discover the most common ones.

To identify either uncorrelated key pairs or negatively correlated key pairs, additional statistics are needed to find the best publishers to index a multi-key continuous query. To clarify this need, consider a key pair ab that has no correlation. For these keys, there are only few publishers in the network that have the potential to publish in the future relevant documents that would contain both a and b, and we cannot locate these publishers by selecting and combining the statistics for key a and key b alone. The conclusion from this is that we consider a pair ab as interesting if both $\hat{P}(A|B)$ and $\hat{P}(B|A)$ are *below* some threshold β within the documents. Extending the above to a key set S with an arbitrary number of keys using the conditional probability in Equation 7, we have to estimate for each key k_0 the value of $\hat{P}(K_0|K_1 \ldots K_{n-1})$. The key set S is of interest if all estimated probabilities are *below* some threshold β.

4.2 The Algorithm USS

In this section we present the algorithm USS (Unique Synopses Storage) that uses single-key synopses already stored in the directory to estimate the publishing behavior for key sets. As the distributed statistics of a publisher p_i regarding a key a contain the synopsis $SYN_i(a)$ (to represent the documents of p_i), it is possible to compute the number of documents containing all keys of a given key set S. For this, we use the intersection operation for multisets as explained in Section 2 as an estimation for the frequency of the whole key set in p_i's collection ($df_i(S)$). Together with prediction techniques presented in [27], the most promising publishers can be selected by applying appropriate scoring functions based on publisher behavior prediction.

Consider a subscriber s subscribing a multi-key continuous query $q = \{k_1, k_2, \ldots, k_n\}$ containing n distinct keys. According to the USS algorithm the following steps are executed:

1. For each key $k_j, 1 \leq k \leq n$ in q, subscriber s contacts the directory node $d(k_j)$ and retrieves the statistical summaries published individually for all keys, including the synopses $SYN_i(k_j)$ produced by publisher p_i.

2. For each publisher p_i appearing in all statistics, s computes an estimation for $df_i(q)$ using synopses intersection techniques, and applies prediction techniques based on time-series analysis to compute a behavior prediction score as described in [27]. This score indicates p_i's potential to provide appropriate published documents in the future.

3. Subscriber s sends the continuous query q to the top-ranked publishers with the highest prediction scores. Only these publishers will index q and in the future notify s about matching documents. Obviously, non-selected publishers will not notify s for published matching documents, since they are not aware of q.

4. Due to publisher churn and dynamics in publishing behavior s has to reposition q by repeating steps 1 to 3 in a periodic way.

The USS approach has the major advantage that it can be performed for all possible key sets and queries in the directory, while the CSS approach, presented in the next section, uses multi-key statistics of judiciously selected key sets, and can only be applied to these *predefined* key set collections. Below, we list some important issues about the USS approach:

- **Higher Network Load:** To exploit the single-key statistics for a given key set, the directory has to send long lists of statistics to the requesting subscriber. In contrast, given multi-key statistics in the directory, only the statistics of the top-ranked publishers have to be retrieved.

- **Inaccuracy:** While both Hash Sketches and KMV synopses allow to estimate the number of distinct values in the intersection of multisets, it is inaccurate to create a synopsis for the intersection that represents the documents containing all keys. In the case of Hash Sketches an estimation for a key set with more than two keys suffers from a significant degrading in accuracy due to its indirect way to compute the sieve formula, whereas the KMV synopses provide better cardinality estimation for multiple set intersections.

- **Prediction Errors:** Considering single-key statistics to predict publisher behavior introduces additional errors. The subscriber has to perform time-series analysis of estimated values thus increasing the probability of prediction errors whereas prediction on accurate values showed very promising results [27].

4.3 The Algorithm CSS

To overcome the problems of algorithm USS, we introduce the algorithm CSS (Combined Synopses Storage) that identifies valuable key combinations, enhances the directory to explicitly maintain these multi-key statistics, and exploits these statistics to improve publisher selection.

We have already discussed in Section 4.1 that uncorrelated or negatively correlated keys are of interest to collect multi-key statistics. Our *CSS* algorithm aims at addressing the following questions: (i) how to decide which key sets are of interest to disseminate additional multi-key statistics; (ii) how to disseminate multi-key statistics of publishers to the directory; (iii) how to exploit the additional statistics to improve publisher selection; (iv) how to deal with multi-key statistics for subsets of the complete continuous query.

4.3.1 Assessing Key-Sets

Given a key set $S = \{k_1, k_2, \ldots, k_n\}$, we employ a deterministic function (e.g., by selecting the lexicographically highest or lowest key) to select one directory node that has to assess the relatedness among the keys of S and be responsible for this key set. To achieve this, the CSS approach uses the following steps to assess a candidate key set S where $d(S)$ is the directory node responsible for the decision:

1. Initially, $d(S)$ contacts all directory nodes responsible for the other keys $k_j \in S$ to retrieve the synopsis $SYN(k_j)$ for k representing all documents containing the key. The answering directory nodes for a key k_j compute locally the synopsis $SYN(k_j)$ by using the union operation over all individual publisher synopses $SYN_i(k_j)$ (see Sections 2.1 and 2.2).

2. Subsequently $d(S)$ computes the intersections among the synopses, to retrieve the estimated cardinality of documents containing all keys ($df(S)$).

3. Using Equation 7, $d(S)$ then computes the conditional probabilities for each key k_j.

Small values for the conditional probabilities show that the occurrence of all keys in the documents are largely independent, meaning that publisher selection decisions can strongly benefit from the existence of available multi-key statistics. Thus, CSS initiates the creation of these additional summaries if the conditional probabilities for all keys $k_j \in S$ are below some threshold β.

To further optimize message traffic between directory nodes we introduce a threshold α, such that if the conditional probability of a key k_j is above α, publisher selection strategy does not have to consider the single-key statistics of that key. The idea behind this is that k_j is contained in *almost all* documents containing the rest of the keys $S \backslash \{k_j\}$, making this key set a candidate for multi-key statistics.

4.3.2 Disseminating Multi-Key Statistics

As soon as a key set has been assessed as a useful candidate that is worth collecting multi-key statistics for, publishers have to learn this fact. Especially in an information filtering scenario, where statistics have to get updated, the continuous process of directory refreshment can be used to disseminate the knowledge about multi-key sets.

Let us assume that a directory node $d(S)$ responsible for a key k identified key set S as useful. According to our algorithm, $d(S)$ is also responsible to maintain the multi-key statistics for the complete key set S. Whenever a publisher p updates its statistics for key k, $d(S)$ can inform the publisher about the new key set. In this way p will compute its local statistics for the set S and disseminate them to the directory node $d(S)$. This procedure does not incur any additional messages compared to the baseline approach described in Section 3.1, since additional statistics can be piggybacked on messages that need to be sent anyway.

4.3.3 Exploiting Multi-Key Statistics

Let us assume that a subscriber s submitting a continuous query q asks the directory nodes responsible for the keys in q to provide the related statistics. Since q is included in the request message, a directory node that stores statistics about a key set S, where $S \subseteq q$, will return the statistics for S together with the single-key statistics it is responsible for. In the following section we explain how to compute the statistics for a multi-key query by combining statistics from subsets available in the directory.

Subsequently, the subscriber collects the multi-key statistics and uses them to perform publisher ranking. Note that the subscriber is based on multi-key statistics, and is thus able to predict the publishing behavior of sources more accurately. Contrary, the baseline algorithm of Section 3.1 performs a separate prediction for each single key.

We also have to mention that if no summaries for the key set (or subsets, see 4.3.4) have been published, the baseline procedure using single-key summaries is still applicable without contacting additional directory nodes.

4.3.4 Combining Multi-Key Statistics

A usual case for continuous queries with more than two keys is that there are no statistics for the full key set S of the query, because the assessment step did not consider the full key set as sufficiently useful. However there might be available statistics for several multi-key *subsets* of S. In this case, by design, all of these subsets can be found by the subscriber by simply asking the directory nodes responsible for the single keys as explained in the previous section.

To give an example, assume that the directory maintains the multi-key statistics of $S_1 = \{a, b, c\}$ and $S_2 = \{b, c, d\}$. Then, for a given five-key query $S = \{a, b, c, d, e\}$, we have several options at hand. In our CSS approach, we propose to select all maximal subsets among the available multi-key summaries. This is efficient in terms of network costs because the entire continuous query will be sent to all single-key directory nodes anyway. But this consideration opens up a space of optimization strategies; Combining such incomparable but mutually related multi-key statistics is reminiscent of the recent work on multidimensional histograms with incomplete information [16], but our setting has the additional complexity of very-high-dimensional key space.

To combine the different multi-key statistics, we weight them depending on the size of the multi-key subset. The scoring function to calculate a publisher score is given by:

$$score_S(p) = \sum_{S_i \subseteq S} |S_i| \cdot predScore_{s_i}(p) \qquad (8)$$

S_i is a subset of the key set S contained in the continuous query, $|S_i|$ is its cardinality, and $predScore_{s_i}(p)$ represents the likelihood that p will produce a document containing S_i in the future. This score is produced using time-series analysis techniques similarly to [27]. Thus, to obtain a publisher score, we sum-up all prediction scores for the subsets S_i that we have received from the directory nodes such there is no S_j with $S_i \subset S_j$. The intuition behind weighting the prediction score with the size of the key set $|S_i|$ is that the prediction score for small subsets dominates the sum. This happens because the number of documents containing all the keys of small key sets is higher, resulting in higher prediction scores. In the next section, we experimentally investigate different scenarios where multi-key statistics for the full continuous query are not available, but a combination of the statistics of subsets is possible.

5. EXPERIMENTAL EVALUATION

In this section we evaluate the USS and CSS algorithms using two real-life corpora with web and blog data. Since it is difficult to find real-life continuous queries except by obtaining proprietary data (e.g., from CNN's news or Springer's journal alert system), we used the popular web queries contained in the *Zeitgeist query-log*, and treated them as subscriptions (e.g., assuming that a user is interested in staying informed about his favorite pop star). We compare the filtering performance of our algorithms with different synopses and against a baseline algorithm that computes publisher scores using single-key frequencies to predict the publishing behavior of information producers. Our prediction mechanisms have an average prediction error of 10%.

5.1 Experimental Setup

Data sets. For the experimental setup we use a recently proposed benchmark [19] designed specifically for use in the evaluation of distributed and P2P settings. Briefly, the benchmark consists of more than 800,000 web documents drawn from the Wikipedia corpus, and an algorithm to distribute the documents among 1,000 publishers with controlled overlap that mimics the real-world behavior. The Zeitgeist query-log contains queries with one, two, and three keys. To investigate filtering quality depending on the conditional probabilities, we use the set of distinct single-keys in all queries to create two-, three, and four-key queries. During query evaluation, a continuous query is indexed in up to 25% of the publishers. We use the concept of publication rounds to simulate the time properties of the published documents; in our experiments a publisher publishes around 400 documents in each round, which could represent the weekly publications of a digital library or news portal.

Our second data set focuses on crawled blog data. Out of the more than one million blogs, we selected about 1,000 blogs that published more than 300 postings each during a time period of three weeks, and assigned them to 1,000 publishers. Again, we used the same Zeitgeist query-log to evaluate the filtering performance, while our query repositioning algorithm was executed to simulate one query repositioning per week.

Quality measure. We use recall to evaluate the filtering performance of our algorithms. Recall in our IF setting is defined as the ratio of the total number of notifications received by subscribers to the total number of published documents matching the subscriptions. In our experiments, we consider the *average recall* over several publication rounds.

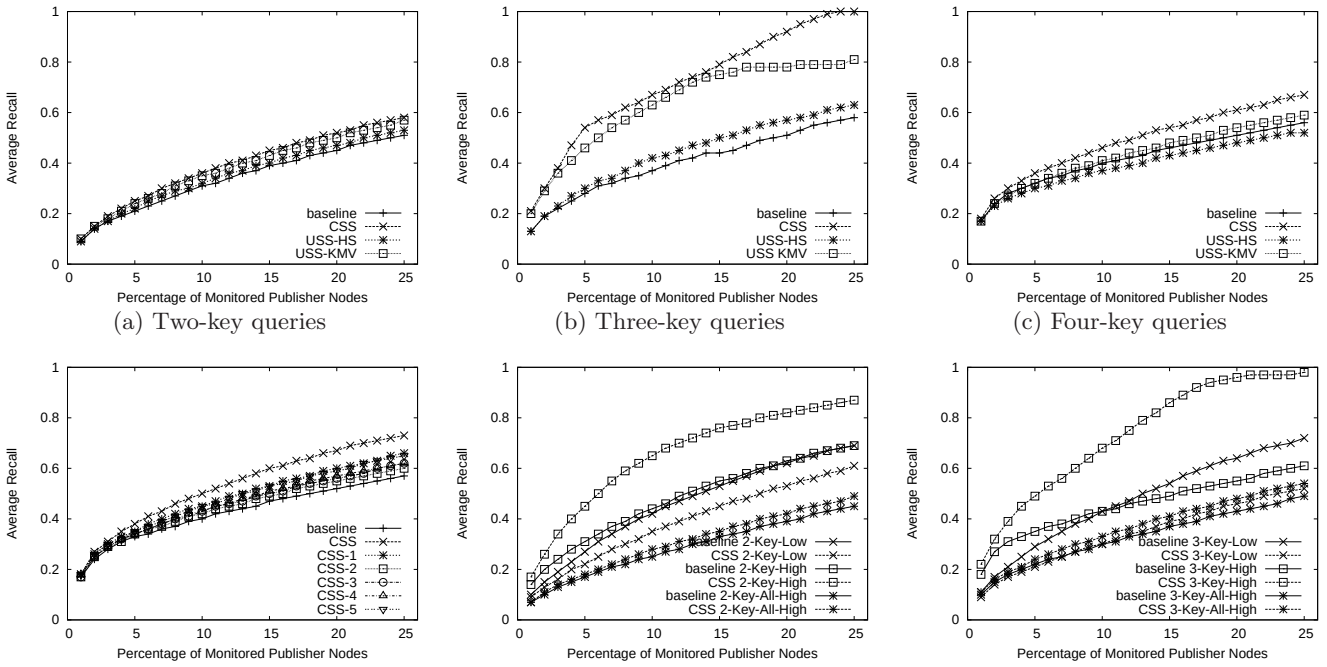
5.2 Experimental Results

Figure 1(a) (resp. 1(b)) shows the filtering quality for all two-key (resp. three-key) continuous queries from our web query-log. We compare a *baseline* publisher selection algorithm based on behavior prediction for single-key statistics [27], with the CSS approach of multi-key statistics maintained in the directory, and with the USS approach using two different synopses (Hash Sketches and KMV synopsis).

The results show the recall improvements obtained by our algorithms. For two-key queries, 24% publishers have to be monitored to reach a recall level of 0.5 in the *baseline* approach, whereas, using the CSS approach, the subscriber only has to monitor 19% of the network. The CSS outperforms both USS approaches, because it offers more accurate and explicit statistics for the key-pairs. Comparing the two USS approaches, the use of KMV synopses slightly improves filtering quality when compared to Hash Sketches. Considering the results for the three-key query set, the improvements are much higher. The CSS approach reaches a recall of 0.79 by subscribing to 15% of all publishers. In contrast, the *baseline* approach reaches a recall of 0.44 by monitoring the same number of publishers. Using this query set, we see a considerable difference between the results of the two USS approaches. The *USS-KMV* approach that uses KMV synopses almost reaches the result quality of the multi-key approach, while *USS-HS* suffers from the inaccuracy of combining Hash Sketches of more than two sets. Recall that KMV synopses uses a direct way to intersect multisets, whereas Hash Sketches reside on an indirect computation that causes loss in accuracy.

To investigate the filtering quality of our approaches for four-key continuous queries, we created queries using the single keys from the real-world query set. Out of the full set of all possible four-key queries, we selected the 500 combinations with the highest number of matching documents in our web collection. Examples of resulting queries are *time war unit world* and *day nation world game*.

In our first experimental series with this query set, we investigated the general filtering quality of our CSS and USS approaches. Figure 1(c) shows that CSS outperforms the *baseline* approach and both USS approaches. When Hash Sketches are used to represent documents the filtering performance is worse even from the *baseline* approach due to the inaccuracy of multiset operations. Contrary, the use of KMV synopses improves the filtering quality because it guaranties better distinct-value estimations for the intersections of multiple sets. Notice that our selected four-key queries do not



(a) Two-key queries (b) Three-key queries (c) Four-key queries

(d) Combining multi-key statistics (e) Effect for two-key queries (f) Effect for three-key queries

Figure 1: Average recall for multi-key queries from *Zeitgeist* on the Wikipedia collection.

fully show the improvement margins of the CSS algorithm, since more selective key sets with less matching documents benefit more from multi-key statistics. The three-key queries in Figure 1(b) are more selective with less matching documents resulting in higher improvements.

Using the set of four-key continuous queries described above, we measured the filtering quality for combining multi-key statistics as proposed in Section 4.3.4. Figure 1(d) illustrates the improvements for several scenarios of available multi-key statistics in comparison to the baseline approach: all four three-key statistics (*CSS-1*), all six two-key statistics (*CSS-2*), two two-key statistics (*CSS-3*), one three-key plus one single-key statistics (*CSS-4*), and one two-key plus one three-key statistics (*CSS-5*). The results show that all combinations improve the *baseline* approach, but cannot reach the filtering quality of the CSS approach.

Figures 1(e) and 1(f) demonstrate the improvements in filtering performance over the *baseline* algorithm, when using our correlation measures for multi-key statistics. To conduct this experiment, we created all possible two- and three-key queries using the single-keys from our *Zeitgeist query-log* and selected three query sets (with a size of 100 each) with the following properties: (i) *Low-Corr* includes those key sets where all keys have a low conditional probability; (ii) *High-Corr* consists of key sets with high conditional probability for *at least one* key; (iii) *All-High-Corr* consists of key sets with high conditional probabilities for all keys. Table 2 shows example three-key queries for the query sets described above (* denotes high conditional probabilities).

key_A, key_B, key_C	$P(A\|BC)$	$P(B\|AC)$	$P(C\|AB)$
british, sport, star	0.13	0.21	0.22
time, face, friend	0.83*	0.25	0.22
nation, unit, world	0.54*	0.47*	0.45*

Table 2: Relatedness of three-key queries.

As already explained, subscribing to key sets where all keys have low conditional probabilities yields to the highest filtering result improvements when monitoring the same percentage of publisher nodes. Thus, when monitoring only 10% of the publishers, *3-Key-Low-Corr* has an recall improvement from 0.44 to 0.65 whereas *3-Key-All-High-Corr* only improves filtering quality from 0.25 to 0.28. Similar results hold for two-key queries leading us to the conclusion that the CSS algorithm has no significant effect for key sets where all keys are highly correlated, while it significantly improves filtering for key sets with low correlations.

Analyzing the results for key sets where one key is highly correlated to all others (*High-Corr*), we observe an smaller improvement compared to unrelated keys. Here, the use of the CSS approach is possible, but we propose an alternative strategy: a high conditional probability for a key k means that there is almost no additional information included in multi-key statistics for the full key sets in comparison to the key set without k. As an example, the multi-key statistics for key set *time face friend* yield to similar filtering results than the multi-key statistics for *face friend* because the conditional probability $P(time|face, friend)$ is high (83% of all documents containing *face* and *friend* also contain *time*).

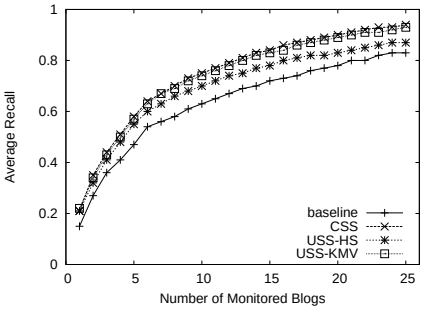


Figure 2: Average recall for blog data.

Figure 2 shows experimental results from our blog data experiment. Our observations lead to similar conclusions with the web data corpus, with CSS outperforming USS and *baseline*. For space reasons, we only present the results for all two-, and three-key queries from the *Zeitgeist* query-log. Again, the KMV synopses perform better than Hash Sketches. Notice that, contrary to the web data collection, a smaller number of blogs have to be monitored to acquire a sufficiently high recall. This happens because all blogs are highly specialized and thus only a small number of them contribute new posts matching the continuous queries.

6. RELATED WORK

In this section, we present related work in the context of information retrieval and filtering, focusing on advances on distributed and P2P settings, and discuss the usage of correlation among keys within these settings.

Recent work on P2P IR is related to earlier research on distributed IR or metasearch engines [17], but this older work assumed a small and static set of document collections among which a query had to choose. In contrast, P2P systems consider a much larger scale and also face high dynamics, which rules out comprehensive and complex statistical models for query routing. The most important routing methods in the literature are CORI [9], the decision-theoretic framework (DTF) [14], and the overlap-aware method [5].

All these methods organize the statistics about nodes, which drive the query routing decisions, on a per-key basis disregarding key correlations. The only recent works that consider key correlations in the context of P2P search are [18] and [23]. [18] considers frequent key combinations in query logs and documents for P2P IR, where [21] proposes a framework for discriminative keys, which includes correlated key combinations; however, it does not give any algorithms for managing the corresponding statistics in a distributed setting and for correlation-aware query routing.

In the area of distributed and P2P IF, new approaches that use a DHT as the routing infrastructure to build filtering functionality have lately been developed. Scribe [22] is a topic-based publish/subscribe system based on Pastry. Hermes [20] is similar to Scribe because it uses the same underlying DHT but it allows more expressive subscriptions by supporting the notion of an event type with attributes. In [25], pFilter uses a hierarchical extension of CAN DHT to filter unstructured documents and relies on multi-cast trees to notify subscribers. Finally, supporting prefix and suffix queries in string attributes is the focus of the DHTStrings system [2], which utilizes a DHT-agnostic architecture to develop algorithms for efficient multi-dimensional event processing. In [27], two systems for exact and approximate information filtering (coined DHTrie and MAPS respectively) are compared where the approximate approach also uses an distributed directory of metadata concerning publishers.

P2P-DIET [15] and LibraRing [26] where the first approaches that tried to support both retrieval and filtering in a single unifying framework. P2P-DIET utilizes an expressive query language based on IR concepts and is implemented as an *unstructured P2P network* with routing techniques based on shortest paths and minimum-weight spanning trees. LibraRing was the first approach to provide protocols for the support of both IR and IF functionality in digital libraries using DHTs. Contrary to LibraRing, in MinervaDL [29] the Chord DHT is used to disseminate and store *statistics or metadata* about the publishers rather than the documents themselves. Avoiding per-document indexing granularity allows to improve scalability by trading recall for lower message traffic. This approximate retrieval and filtering approach relaxes the assumption of potentially delivering notifications from every producer and amplifies scalability. The work in this paper shares the concept of approximate IF with the MinervaDL system and uses IF techniques similar to the ones described there.

7. CONCLUSIONS

In this paper, we presented two algorithms, coined USS and CSS, that capture and exploit statistics about correlated key sets to improve the filtering performance in approximate IF scenarios. To improve efficiency we also used Hash Sketches and KMV synopses to compactly represent publisher content. Our experimental studies showed that algorithm CSS outperforms the competitors and achieves an average recall of around 70% while monitoring only 10% of the publishers. Additionally, the usage of the more accurate KMV synopses, instead of Hash Sketches, enables the usage of USS also for multi-key queries, since its filtering effectiveness is greatly improved. Finally, we observed that, multi-key queries with keys that are uncorrelated show the highest gains in terms of recall.

8. REFERENCES

[1] K. Aberer. P-Grid: A Self-Organizing Access Structure for P2P Information Systems. In *CoopIS*, 2001.

[2] I. Aekaterinidis and P. Triantafillou. Internet Scale String Attribute Publish/Subscribe Data Networks. In *CIKM*, 2005.

[3] R. Agrawal, T. Imielinski, and A. N. Swami. Mining Association Rules between Sets of Items in Large Databases. In *SIGMOD*, 1993.

[4] Z. Bar-Yossef, T. S. Jayram, R. Kumar, D. Sivakumar, and L. Trevisan. Counting Distinct Elements in a Data Stream. In *RANDOM*, 2002.

[5] M. Bender, S. Michel, P. Triantafillou, G. Weikum, and C. Zimmer. Improving Collection Selection with Overlap Awareness in P2P Search Engines. In *SIGIR*, 2005.

[6] K. S. Beyer, P. J. Haas, B. Reinwald, Y. Sismanis, and R. Gemulla. On synopses for Distinct-Value Estimation Under Multiset Operations. In *SIGMOD*, 2007.

[7] B. H. Bloom. Space/Time Trade-offs in Hash Coding with Allowable Errors. *CACM*, 1970.

[8] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher. Min-Wise Independent Permutations. *JCSS*, 2000.

[9] J. P. Callan, Z. Lu, and W. B. Croft. Searching Distributed Collections with Inference Networks. In *SIGIR*, 1995.

[10] G. Cormode and M. N. Garofalakis. Sketching Streams Through the Net: Distributed Approximate Query Tracking. In *VLDB*, 2005.

[11] M. Durand and P. Flajolet. Loglog Counting of Large Cardinalities (Extended Abstract). In *ESA*, 2003.

[12] M. Fang, N. Shivakumar, H. Garcia-Molina, R. Motwani, and J. D. Ullman. Computing Iceberg Queries Efficiently. In *VLDB*, 1998.

[13] P. Flajolet and G. N. Martin. Probabilistic Counting Algorithms for Data Base Applications. *JCSS*, 1985.

[14] N. Fuhr. A decision-theoretic approach to database selection in networked ir. *ACM TOIS*, 1999.

[15] S. Idreos, M. Koubarakis, and C. Tryfonopoulos. P2P-DIET: An Extensible P2P Service that Unifies Ad-hoc and Continuous Querying in Super-Peer Networks. In *SIGMOD*, 2004.

[16] V. Markl, N. Megiddo, M. Kutsch, T. M. Tran, P. J. Haas, and U. Srivastava. Consistently Estimating the Selectivity of Conjuncts of Predicates. In *VLDB*, 2005.

[17] W. Meng, C. T. Yu, and K.-L. Liu. Building Efficient and Effective Metasearch Engines. *ACM Computing Surveys*, 2002.

[18] S. Michel, M. Bender, N. Ntarmos, P. Triantafillou, G. Weikum, and C. Zimmer. Discovering and Exploiting Keyword and Attribute-Value Co-occurrences to Improve P2P Routing Indices. In *CIKM*, 2006.

[19] T. Neumann, M. Bender, S. Michel, and G. Weikum. A Reproducible Benchmark for P2P Retrieval. In *ExpDB*, 2006.

[20] P. R. Pietzuch and J. Bacon. Hermes: A Distributed Event-Based Middleware Architecture. In *DEBS*, 2002.

[21] I. Podnar, M. Rajman, T. Luu, F. Klemm, and K. Aberer. Scalable Peer-to-Peer Web Retrieval with Highly Discriminative Keys. In *ICDE*, 2007.

[22] A. I. T. Rowstron, A.-M. Kermarrec, M. Castro, and P. Druschel. SCRIBE: The Design of a Large-Scale Event Notification Infrastructure. In *Networked Group Communication*, 2001.

[23] G. Skobeltsyn, T. Luu, I. P. Zarko, M. Rajman, and K. Aberer. Web Text Retrieval with a P2P Query-Driven Index. In *SIGIR*, 2007.

[24] I. Stoica, R. Morris, D. R. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. In *SIGCOMM*, 2001.

[25] C. Tang and Z. Xu. pFilter: Global Information Filtering and Dissemination Using Structured Overlay Networks. In *FTDCS*, 2003.

[26] C. Tryfonopoulos, S. Idreos, and M. Koubarakis. LibraRing: An Architecture for Distributed Digital Libraries Based on DHTs. In *ECDL*, 2005.

[27] C. Tryfonopoulos, C. Zimmer, G. Weikum, and M. Koubarakis. Architectural Alternatives for Information Filtering in Structured Overlays. *IEEE Internet Computing (IC)*, 2007.

[28] T. Yan and H. Garcia-Molina. The SIFT Information Dissemination System. *ACM TODS*, 1999.

[29] C. Zimmer, C. Tryfonopoulos, and G. Weikum. MinervaDL: An Architecture for Information Retrieval and Filtering in Distributed Digital Libraries. In *ECDL*, 2007.