Efficient Query Routing by Improved Peer Description in P2P Networks

Wai Gen Yee, Linh Thai Nguyen, Dongmei Jia
Department of Computer Science
Illinois Institute of Technology
Chicago, IL 60616 USA
+1-312-567-5330
{waigen, linhnt, jia}@ir.iit.edu

Ophir Frieder
Department of Computer Science
Georgetown University and IIT
Washington, D.C. 20057 USA
+1-202-687-2165
ophir@cs.georgetown.edu

ABSTRACT
Peer-to-peer file-sharing systems commonly use the set-of-terms model to describe succinctly a peer's shared file set: the union of the terms in the share files. This information is used to guide query routing decisions. The problem with this model, however, is that it falsely suggests term co-occurrences that do not exist in any single file. Consequently, queries get routed erroneously to peers that have no matching files, wasting network and computation resources in the process. We reduce the amount of co-occurrence errors by partitioning each peer's file set and representing the peer as several file partitions instead of one. Experimental evidence demonstrates that it is possible to reduce the network traffic between neighbors by up to 60% at virtually no cost.

Keywords
Peer-to-peer, file-sharing, collection description, routing

1. INTRODUCTION
In weakly structured peer-to-peer file-sharing systems (e.g., Gnutella), a peer makes query routing decisions by judging the likelihood that a neighbor's shared content contains matching results. The peer compares the query to a locally stored description of each neighbor's shared content. If the description does not preclude the possibility of a match, then the query is routed to the corresponding neighbor.

A common way of implementing a peer's description is as a "term set," comprised of the union of the terms in each of its shared files (or filenames in the case that the files are binary [3]). Queries are matched conjunctively; if a term set contains all query terms, it is routed to the corresponding peer. A file matches a query if its filename contains all query terms.

The term set model, however, lacks precision in describing content and may lead to queries being erroneously routed to neighbors with no matching results. Such erroneous routing wastes both network and peer resources. For example, if a peer shares two files, named "world" and "cup," respectively, its term set would be {world, cup}. In this case, the query "world cup" will be routed to it, even though none of its files are actually matches. We refer to this phenomenon as the *false co-occurrence* problem referring to the fact that the term set falsely suggests that "world" and "cup" co-occur in some filename.

False co-occurrence is a significant problem in real-world P2P file-sharing systems. In our experiments with data collected from the Gnutella network, well over 50% of the queries routed to an average peer do not match any of that peer's local files.

To address the false co-occurrence problem, we propose partitioning the peer's shared file collection and creating a term set for each partition – each peer is represented by a collection of term sets instead of just one. A partitioned term set is a more precise description of a shared set of files because it leads to fewer co-occurring errors and ultimately less network traffic. Returning to the example above, we could create two partitions for the peer, one for each file, resulting in the descriptor: {{world}, {cup}}. Because the query "world cup" matches neither term set in the descriptor, it would not be routed to the corresponding peer.

Each partition represents a subset of a peer's file collection; thus we expect it to be more precise a representation. Our main contribution explores the allocation of files to the partitions. We contrast our work to previous work on the representation of arbitrary collections in networks that partitions files randomly (e.g., [29]) as well as previous work in meta-search engines that partitions files semantically [12][30]. Unexpectedly, none of these techniques work very well in our application so we propose a novel alternative that improves on these by up to 55%.

Second, we consider the cost in terms of space of maintaining several data structures to represent a single collection, which may be a factor particularly in networked environments. We consider the cost/benefit tradeoffs that are incurred by fixing the size of the representation – the cost decrease using fixed-size representations is at most 50% in our experiments. We also propose a technique that dynamically balances cost and benefit. This dynamic technique is able to compute a fixed-size representation that reduces cost to within 8% of manually tuned representations.

Infoscale'08, June 4–6, 2008, Vico Equense, Italy.

2. RELATED WORK

Distributed hash tables (DHTs) (e.g., [17][31]) are an efficient approach to search in reliable networks. Basic DHTs, however, only allow search for single keys. Since general information retrieval systems allow multi-term queries, distributed inverted lists based on DHTs were devised [5][11][18]. In such systems, each term is associated with the list of documents identifiers (doc-ids) containing it. The DHT maps each of these lists to nodes based on its associated term. *n*-term user queries retrieve n doc-id lists, and the intersection of these lists is the set of identifiers of the documents that contain all of the terms. The problem addressed here is in reducing the cost of retrieving all the lists, each of which may be very long.

The work on search with DHTs is distinct from ours for several reasons. First, the application of DHTs requires some system stability, which most P2P file-sharing applications do not exhibit [19]. Second, they assume that the shared data constitute an integrated collection, whereas in P2P file-sharing systems, each peer is considered a separate collection, requiring its own search mechanism. The specific problem they address (i.e., the cost of retrieving and intersecting several large lists) is distinct from ours (i.e., representing a single peer's shared collection).

A more general problem of representing a collection of shared data has been considered in the information systems community [5][8][9][20][21][22][23]. These techniques include using hash sketches, vector space models, language models, and approximate collection samples, which are then used to guide whether queries should be issued to the corresponding data sources. However, as these representations are only approximations, query routing choices are subject to *both* false-positive and false-negative errors. Our goal is to reduce the false-positive routing errors without introducing any false negative ones. Furthermore, our solution of partitioning files can be combined with the aforementioned representation techniques to improve their precision.

Specific routing techniques have been devised for the P2P environment [12][24][25][26]. The focus of [12][24] is on index propagation and update management, but use collection representations similar to those mentioned in the previous paragraph. Specifically, because the problem of propagating and updating indices are largely orthogonal to collection representation, it is straightforward to apply these techniques to our collection representation technique. (For example, the proposed techniques for representing multiple collections as one to create a collection representation hierarchy can be straightforwardly applied to our representation techniques.)

[25][26], on the other hand, focus on representing XML documents and queries either as an adjacency list [25] or as a matrix [26]. However, these works do not address the problem of removing the false co-occurrence problem while controlling the complexity of collection representation, which makes them orthogonal to our problem. Their solutions are also based largely on the graph structure of XML documents.

Our work improves the precision of the peer descriptor without introducing any false-negative routing errors by reducing the number of false co-occurrences implied by a peer's descriptor.

3. QUERY PROCESSING AND ROUTING SPECIFICATION

In P2P file-sharing systems, each peer P shares a collection R of files. Each file $F_j \in R$ is represented by a *file descriptor* (e.g., a filename), denoted $D(F_j)$, which is a set of "terms."

Let T be the union of all terms of all file descriptors of files in R (i.e., $T = \cup_{Fj \in R} D(F_j)$). We refer to T as the *peer descriptor* of P or the *term set* of R.

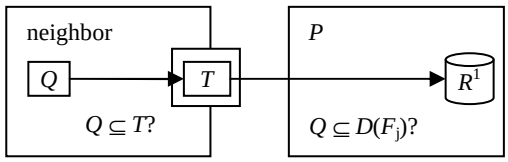
A query Q is also a set of terms. A query Q that is routed to a peer P is compared with all files in $F_j \in R$. Queries are processed conjunctively, so Q *matches* F_j if Q is a subset of the $D(F_j)$ (i.e., $Q \subseteq D(F_j)$) [3]. In the event of a match, $D(F_j)$ and P's id are returned to the client who issued Q, which uses this information to decide on whether to download the associated file.

In practice, a peer only maintains collection information about its immediate neighbors. This information is updated at regular periods or whenever a peer joins the network.

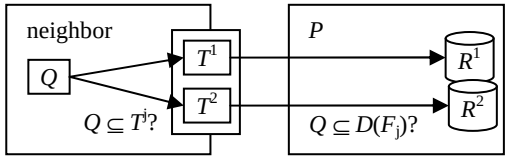
Each query Q is initialized with a time-to-live of 3 to 7 and flooded in the network until the last hop. The last hop routing decision to P is made by comparing Q to T. T matches Q if T contains Q (i.e., if $Q \subseteq T$). This match *suggests* that R contains a file F_j that matches Q. A non-match *guarantees* that no file in R matches Q.

This last-hop routing design is a reasonable cost/benefit trade-off: maintaining collection information beyond immediate neighbors is complex and expensive [12], and, in any case, the majority of network traffic happens during the last hop.

Because peer descriptors are only used on the last hop, without loss of generality, our model consist of a single peer P to which an abstract "neighbor" routes queries based on peer descriptor T as shown in Figure 1a



a. Basic routing



b. Routing with a 2-partitioned collection.

Figure 1. Basic (a) and partitioned collection (b) routing.

4. PROBLEMS WITH TERM BASED QUERY ROUTING

A *co-occurrence error* occurs between Q and P if Q matches T, but no file $F_j \in R$ matches Q:

$$Q \subseteq T \text{ and } \neg\exists\, F_j \in R \text{ s.t. } Q \subseteq D(F_j).$$

In this case, Q is *erroneously routed* to P, wasting network bandwidth and computation resources at S. We refer to routing decisions due to co-occurrence errors as a type of *false positive* error because T_i falsely suggests that routing Q to S_i would result in at least one positive match. Note that *false negatives* cannot occur in the given model.

Co-occurrences errors occur because T generally suggests the existence of file descriptors for files in R that do not exist. The number of unique queries that T matches is exponential in $|T|$. T theoretically matches $2^{|T|}$ - 1 unique queries, but there are far fewer than this number of files in R in practice. Reasonable values for $|T|$ and $|R|$ are 500 and 100, respectively, which suggests the potential impact of this problem.

We use the following ratio to measure the degree of false co-occurrences that exist in a description T of a collection R:

$$G(R) = 1 - N_c / (|\mathcal{P}(T)| - 1), \quad (1)$$

where N_c is the total number of non-empty queries that can match some file in R and $\mathcal{P}(T)$ denotes the power set of T (i.e., the set of all possible queries that can match T). The second term on the right side of Equation 1 represents the proportion of non-empty queries that match both P's descriptor, T, and at least one file in R. One minus this value is the proportion of queries that erroneously match T.

N_c can be expressed as the union of power sets of all $D(F_j)$, where $F_j \in R$:

$$N_c = |\cup_j \mathcal{P}(D(F_j))| - 1, \text{ where } F_j \in R.$$

By definition, $|\mathcal{P}(D(F_j))|$ is equal to $2^{|D(Fj)|}$, but a closed form expression for N_c is not possible in general due to the possible overlaps in $\mathcal{P}(D(F_j))$ for various j.

Our goal is to minimize $G(R)$ by partitioning R. Intuitively, partitioning R reduces $|\mathcal{P}(T)|$, thereby reducing $G(R)$, which we explain in more detail below.

5. SOLVING CO-OCCURRENCE ERRORS BY PARTITIONING THE FILE SET

We reduce the degree of false co-occurrences that occur in T by K-partitioning collection R into $R^1, R^2, \ldots, R^K$ with corresponding term sets $T^1, T^2, \ldots, T^K$. A peer consequently has K term sets $T^1, T^2, \ldots, T^K$ representing its collection instead of one.

Using partitioned descriptions slightly changes the way that queries are routed. A neighbor routes Q to P if Q matches T^j for any $1 \le j \le K$ as shown in Figure 1b. Although partitioned descriptions increase routing decision overhead by a factor K, this cost is trivial compared with the cost of erroneous routing. However, if desired, it is straightforward to apply the cost-reducing techniques covered in [12] to our approach.

By partitioning the collection, the degree of false co-occurrences for the partitions $R^1, R^2, \ldots, R^K$ becomes

$$G^K(R^1, R^2, \ldots, R^K) = 1 - N_c / (|\cup_j \mathcal{P}(T^j)| - 1). \quad (2)$$

In this equation, the denominator on the right side describes the number of non-empty queries that match at least one of $T^1, T^2, \ldots, T^K$. This expression suggests the positive effect that partitioning R has on the degree of false co-occurrences. Specifically, if we compare the denominator from Equation 2 to that of Equation 1, we see that it must be the case that $G(R) \ge G^K(R^1, R^2, \ldots, R^K)$ because $|\cup_j \mathcal{P}(T^j)| \le |\mathcal{P}(T)|$.

The extreme cases where $K = 1$ or $K = |R|$ are illustrative. When $K = 1$, $G(R)$ clearly equals $G^1(R^1)$. The case where $K = |R|$ is the case where one partition is created for each file (i.e., $T^i = D(F_i), \forall F_i \in R$). In this case, the minimum number of queries matches the peer descriptor of P; equivalently, all matched queries match at least one file in R. The following two results formalize the performances of these two extreme cases.

Lemma 1: When all files are in a single partition, the descriptor matches the maximum number of queries.

Proof: We prove the lemma by contradiction. Assume there is a solution with $K > 1$ such that a query Q matches term set T^i of this solution but does not match term set T of the $K = 1$ solution. This is a contradiction because $T^i \subseteq T$: all queries that match T^i must also match T. □

Lemma 2: When there is one file per partition, all queries that match the (partitioned) descriptor of P match at least one file in R.

Proof: We prove the lemma by contradiction. Assume that there exists some query Q such that Q matches some T^i but does not match any file $F_j \in R$. However, because $T^i = D(F_j)$ for some $F_j \in R$, it must be the case that Q matches F_j.□

Correspondingly, the degree of false co-occurrences with the one-descriptor-per-partition solution is zero:

$$\begin{aligned} &G^K(R^1, R^2, \ldots, R^{|R|}) \\ &= 1 - (|\cup_j \mathcal{P}(D(F_j))| - 1) / (|\cup_j \mathcal{P}(T^j)| - 1) \\ &= 1 - (|\cup_j \mathcal{P}(D(F_j))| - 1) / (|\cup_j \mathcal{P}(D(F_j))| - 1) \\ &= 0. \end{aligned}$$

Increasing K to the limit trivially eliminates false co-occurrences, but is an unscalable solution, considering that some peers share hundreds or thousands of files. The solution is to allow a user-tunable K number of partitions. Thus, our goal is to K-partition R to minimize $|\cup_j \mathcal{P}(T^j)|$ for a given K value. We consider the framework of effective partitioning techniques irrespective of K in Section 6 and then consider the determination of practical values of K in Section 7.

6. EXPERIMENTAL RESULTS

We study the effectiveness of partitioning by applying it to data we collected from the Gnutella network during the Spring of 2007 using our IR-Wire data logging tool [15]. For our experiments, we randomly selected 65,000 queries and 50 random peers that share from 100 to 500 files each. We ran the same experiments with both smaller (50 to 100 files per peers) and larger (1000 to 5000 files per peer) data sets, which revealed

little relative difference; therefore, the presented results are representative.

As done in commercial P2P file-sharing systems, we use a *Bloom filter* to represent a partition of files [2]. A Bloom filter is a fixed length bitmap that compactly represents set membership. A Bloom filter is initially all zeroes when the set is empty. When an item (i.e., a term of a file descriptor in this case) is put into the set, the bit (or set of bits) corresponding to that item is set to one in the Bloom filter. Checking if the item might be in the set using a Bloom filter requires checking if its corresponding bits are set to one.

The use of Bloom filters (and hash-mapped bitmaps in general) to represent a set makes possible *collision errors*, which lead to a type of false positive error distinct from those caused by co-occurrence errors. A collision error is defined as the case where two items set the same bit(s) in the Bloom filter. If one of these items is inserted into the set, a search for the other item using the Bloom filter will return a positive result even if the item is not in the set. Bloom filters, however, do not allow false negative errors.

The number of bits used to represent an item sets in a Bloom filter influences its rate of collision errors (as explained in [2]). We use a single bit to represent each item because this is done in practice [1] and results in fewer collision errors as the number of inserted items increases [2].

Our Bloom filters are created with an MD5–based hash function as done in previous work (e.g., [12][27]). The size of each Bloom filter is 64KB, as it is in the Gnutella network [1].

For each server P, we record the following:

- Q_f – the number of queries routed to P.
- Q_m – the number of queries that match at least one file in R.
- Q_c – the number of queries routed to P due to collision errors with the Bloom filter(s). These queries contain at least one term not found in T.
- Q_d – the number of queries routed to P due to co-occurrence errors. We define Q_d as $Q_f - Q_c - Q_m$. By this definition, Q_d may be under-reported as it may overlap with Q_c. However, the magnitude of overlap should be small as we do not anticipate many collisions.

Our main cost metric is Q_f. Because the query matching technique does not admit any false negative errors, the lower the Q_f, the better. We report values averaged over all 50 peers in our test set.

6.1 Partitioning Technique

We partition R using K-means with random centroids and cosine similarity as the basis for distance between a file descriptor and a centroid [4]. This method is well-understood and a common baseline for partitioning performance in information systems.

To review, K-means initially creates K centroids in space (randomly in our experiments) that form the centers of disjoint clusters. It then iteratively assigns each object that is to be clustered to the centroid to which it is closest (defined by cosine similarity in our experiments). After all objects have been assigned, each centroid is recomputed as the average position of the objects that have been assigned to it. The process of object assignment and centroid computation is repeated until the position of all K centroids stabilizes – that is, until they do not change beyond a user-defined threshold. Equivalently, K-means repeats until no object changes cluster membership from the previous iteration.

Cosine distance between term sets D_1 and D_2 is defined as one minus the cosine similarity of D_1 and D_2:

$$L_{\cos}(D_1, D_2) = 1 - V(D_1) \bullet V(D_2) / (\|V(D_1)\|\|V(D_2)\|).$$

In this equation, $V(D_i)$ is D_i's representation as a term frequency vector, $\bullet$ is the dot product, and $\|V(D_i)\|$ is the length of $V(D_i)$. We also tried more complex distance functions (e.g., the squared Jensen-Shannon divergence [7]), but without significantly different results.

In the case of a tie in cosine distance, we use the term set size of the partition for tie-breaking. A descriptor is assigned to the partition that has the fewest terms. Tie-breaking is particularly important during beginning phases of partitioning as this is when it is most likely that the descriptor has zero cosine similarity with every partition.

We assign the descriptor to the partition with the minimum term set size to encourage the creation of partitions that are as small as possible with as little size variation as possible. We discuss the motivations behind these design goals in later sections.

6.2 Varying the Number of Partitions

We compare the performance of cosine-based partitioning to random partition generation with varying K. Recall in Section 1 our mention of how previous work either used semantic or random partitioning of data when creating collection descriptors. Our use of cosine-based and random partitioning correspond to these approaches.

In Figure 2, we show the performance of cosine and random on cost (Q_f) with increasing K. We also include the minimum Q_f (opt), which corresponds to the number of queries that match some file $F \in R$. We will discuss ΔM in Section 6.2.1.

The minimal cost is 160 queries, whereas the base cost is ~600 queries. This means that approximately 75% of queries are erroneously routed to peers.

Cosine steadily decreases cost from 608 to 495 queries (~18%) as K increases from 1 to 5 partitions. Unexpectedly, random partitioning is more than twice as effective as cosine partitioning over the same interval, decreasing cost by about 44%.

The poor performance of cosine partitioning is due to the partition's physical characteristics, which are caused by the behavior of cosine distance. Objects in a partition created by cosine partitioning have maximum similarity with minimal dissimilarity (as defined by the cosine distance metric). On the other hand, partitions generated randomly have arbitrary similarity. Thus, partitions created by cosine partitioning have less overlap and consequently a lower average size than those of random partitioning as shown in Figure 3 and Figure 4. Low overlap is generally a good characteristic of partitions and is good

in our case as well. However, another characteristic of cosine partitioning makes it worse than random.

The problem with cosine partitioning is that it places all "semantically similar" descriptors in the same partition without regard to partition size. Furthermore, larger partitions contain a larger variety of terms, increasing their similarity to the remaining, unassigned descriptors. Ultimately, the large partitions get even larger.

Although cosine distance also measures dissimilarity between a partition and a descriptor, dissimilarity is unlikely to be a factor because of the skew in term distributions and the small sizes of the descriptors. Therefore, regardless of the degree of dissimilarity, a small amount of similarity is likely to be enough to determine the assignment of a descriptor to a partition.

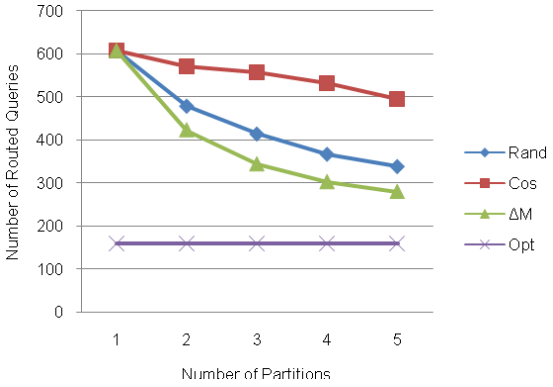


Figure 2. Number of queries routed to P (Q_f) with various partitioning techniques over various numbers of partitions.

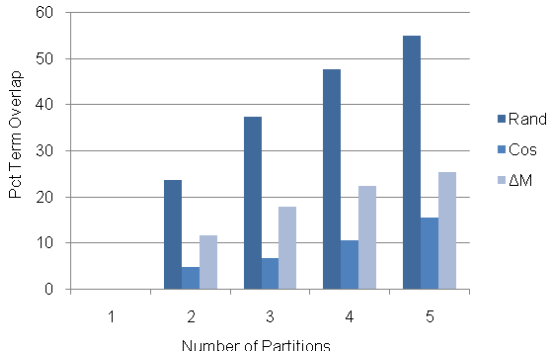


Figure 3. Average percentage overlap between partitions.

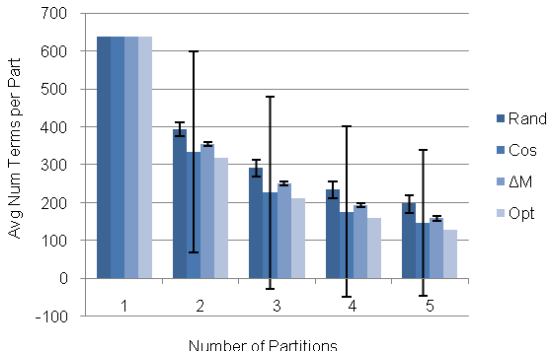


Figure 4. Average number of unique terms per partition. The black bars indicate standard deviations.

Another consequence of the behavior of cosine partitioning is that size variance in solutions is high, as shown in Figure 4. Cosine partitioning tends to create solutions where most descriptors are assigned to very few partitions.

Large partitions must be avoided when creating partitions for routing. Consider the case where no partitions overlap. Then, the number of unique term combinations that match this solution is approximated by the following expression:

$$\sum_{i=1}^{K} 2^{|T^i|}.$$

This expression is minimized when $|T^i|$ is uniform (i.e., $|T^i| = |T| / K$, where $1 \leq i \leq K$) and maximized when $|T_i|$ has high variance (e.g., when $|T^i| = |T|$ for some i). Variance in $|T^i|$, therefore, matches more queries and leads to more routing errors. Furthermore, as suggested in Lemma 1, large partitions approximate worst-case performance.

Random partitioning allows partitions to overlap, leading to larger partitions on average. Randomly generated partitions are as much as 33% larger than cosine-generated partitions on average ($K = 5$ case). However, this does not make up for the fact that cosine generated partitions have a size variance that is 8 times greater than that of randomly generated partitions.

As the number of descriptors grows, however, so does the average size of each partition. In time, random partitioning performance will approach that of cosine partitioning as all terms will be represented in all random partitions. Partition size variance will decrease, but average partition size will be maximized.

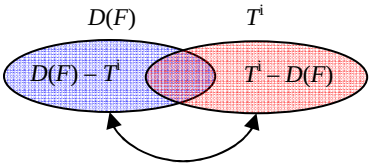
The general problem with cosine partitioning is that it considers descriptor similarity with partitions. Considering the skew in practical term distributions, this leads to high variance in partition size. The general problem with random partitioning is that random partition sizes increase arbitrarily regardless of their contents. To solve the problems of both of these solutions, we propose *difference-based* partitioning.

Difference-based partitioning considers how different a descriptor is to a partition without considering how similar they are when making assignment decisions. By avoiding a similarity comparison, it avoids the problem that cosine partitioning has with size variance. By considering difference, it avoids the problem that random partitioning has of uncontrolled growth.

6.2.1 Difference-Based Distance

The number of false co-occurrences created by assigning file F to R^i can be computed by comparing $D(F)$ with T^i. All of the terms that are in $T^i - D(F)$ falsely co-occur with all the terms that are in $D(F) - T^i$ as shown graphically in Figure 5. The falsely co-occurring term set is computed by the cross product of $(T^i - D(F))$ and $(D(F) - T^i)$. We define the cardinality of this set as the cost of assigning F to R^i, which is computed by

$$\Delta M(D(F), T^i) = |D(F) - T^i| \times |T^i - D(F)|. \qquad (3)$$



False co-occurrences by $(D(F) - T^i) \times (T^i - D(F))$

Figure 5. The false co-occurrences introduced by adding F to R^i are caused by their non-overlapping term sets.

The design of ΔM encourages the creation of partitions with both small average size and small size variance. We can see this by considering its behavior in containment and non-containment conditions between pairs of term sets (i.e., a descriptor and a partition). First, if one descriptor properly contains another, then ΔM, but not cosine, implies that combining them has zero cost. This is the correct behavior as the set of term combinations of the containing term set subsumes that of the contained set. Therefore, combining these two term sets into one introduces no false co-occurrences. This behavior has the tendency to create smaller partitions on average because it avoids partial-containment assignments.

If there is only a partial overlap between term sets, then the ΔM focuses on the difference, whereas cosine focuses on both the difference and the similarity. This focus on the difference has the tendency of assigning descriptors to smaller partitions, which, by expectation, leads to fewer false co-occurrences. This has the effect of reducing the size variance in the final partitioning solution.

We re-ran our experiments, replacing cosine distance with ΔM. Our results, shown in Figure 2 through Figure 4, show that ΔM outperforms both cosine and random partitioning, reducing Q_f by 53% versus 18% and 44% for the other two, respectively. The reason for this performance improvement is clear from the results shown in Figure 4: partition sizes are on average smaller than those of random partitioning and have less variance than those of cosine partitioning.

7. CONTROLLING COSTS: REDUCED-SIZE BLOOM FILTERS

The cost of our proposed solution is a function of K. Larger K values increase routing accuracy but at the expense of more maintenance and transmission cost. We propose to fix cost by fixing the number of bits used to encode the K Bloom filters that represent a peer descriptor; specifically, we fix it at 64KB regardless of K. In doing this, we introduce a tradeoff. Increasing K decreases routing errors due to co-occurrence errors as described in Section 5. However, it also increases the rate of routing errors due to hash collisions; as each partition is allocated fewer bits, the rate of collision errors increases. The challenge is to identify a K that balances these two effects to minimize the overall number of routing errors.

Let N_B be the total number of bits used in the Bloom filter encoding of the peer descriptor T, and let $N_t = |T|$. The probability of a collision in the Bloom filter encoding of T is [2]:

$$\Pr(collision) = 1 - e^{-N_t / N_B}. \qquad (4)$$

We expect that fixing the size Bloom filter is a reasonable approach because N_B is large compared to N_t in practice. Therefore, the probability of a collision is low. Let N_B^i be the number of bits available for the Bloom filter encoding of partition T^i, which contains N_t^i terms. In our data set, a peer's collection contains at most 4,000 unique terms, encoded in 64KB worth of bits. $\Pr(collision\ in\ R^i \mid N_B^i = 64KB / K$ and $N_t = 4{,}000)$ is 0.008 when $K = 1$ and increases at a rate of 0.007 for practical values of K (i.e., $K \leq 20$). This low collision rate gives us the flexibility to increase K.

Two problems arise that complicate increasing K arbitrarily. First, N_t^i decrease more slowly than N_B^i. (Recall the results from Figure 3). Due to overlapping term sets, average N_t^i / N_B^i increases with K and therefore so does the collision rate of each partition.

In addition, the variance in partition sizes increases with K. (Recall the results from Figure 4.) The increased size variance and the increased overlap mean that the ability of partitioning to reduce co-occurrence errors decreases with K. Our goal is to find an optimal K that minimizes Q_f, where the marginal increase in collision errors Q_c is equal to the marginal decrease in co-occurrence errors Q_d.

7.1 Experimental Setup

Since each partition in the optimal solution contains the same number of unique terms, we assign the same number of bits to each partition's Bloom filter: $N_B^i = N_B / K$. Therefore, reducing the range of the hash function $\mathbf{H}$ is straightforward:

$$\mathbf{H}' = \mathbf{H} \bmod N_B^i,$$

where N_B is the number of bits in the original Bloom filter. In addition, each peer descriptor that is transmitted to a neighbor also contains its K value so that the neighbor knows how to parse it for individual Bloom filters.

7.2 Performance of Reduced-Size Bloom Filters

In Figure 6, we plot the performance of reduced-size Bloom filters with different values of K. We break down the performance of ΔM in terms of Q_f, Q_c and Q_d. As expected, as K increases, Q_c increases and Q_d decreases. The rate of decrease in Q_d decreases, however, while the rate of increase in Q_c is linear in our K range, which includes all practical values of K. These trends result in a net increase in Q_f beginning at $K = 8$.

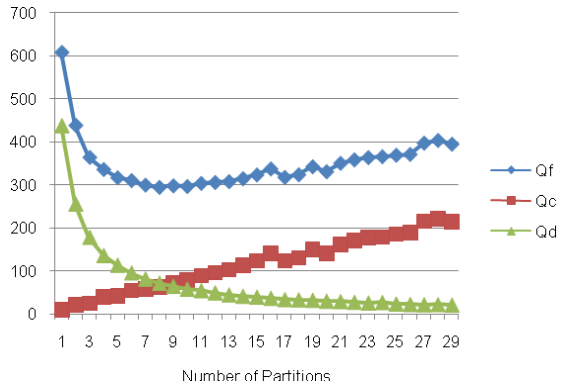


Figure 6. Error breakdown using reduced-size Bloom filters.

7.3 Incremental Collection Partitioning

Although setting K to 8 yields the optimal average performance in our experiments, we cannot generalize the performance of this K value over all peers and all collections. The optimal K value for a particular peer may be greater or less than 8 depending on its term distribution over its file descriptors.

We propose a way of dynamically approximating a local peer's optimal K, balancing the slowing decrease in Q_d with the increase in Q_c as K increases. Our technique uses the expectations that the rate of increase in Q_c is non-decreasing in K and the rate of decrease in Q_d is non-increasing in K. (Both of these expectations can be demonstrated analytically.) Given this behavior, when Q_f begins to increase with K, it is guaranteed not to decrease with increasing K. Therefore, the optimal K is some smaller value.

We propose to increase incrementally K until the rate of decrease in Q_d is below a given threshold, U. Before this threshold, the decrease in Q_d is expected to compensate for the increase in Q_c, but not after it. We use a fixed threshold because we expect Q_c to be linear for our range of K.

During each iteration of our technique, one partition in the set of partitions is *split* in two using ΔM partitioning. If an analysis of the split reveals that it decreases Q_d by a minimum threshold, then the split is accepted and another split is considered. Otherwise, the split is undone at the process stops.

There are two criteria for picking the partition to split:

1. The partition that has been involved in the fewest splits is split first.
2. In the event of a tie, the partition with the most unique terms is split first.

Criterion 1 effects round-robin partition selection by avoiding splitting partitions that have either been either split or created most recently. Because in the optimal solution, N_t^i is equal for all i, each split should create "sibling" partitions with a similar number of unique terms, so splitting the same partition twice consecutively should not be necessary anyway except for degenerate or boundary conditions. In other words, we expect Criterion 1 to be less of a factor in determining which partition to split next than Criterion 2.

Criterion 2 splits the partition with the most terms first because the potential benefit of splitting these partitions is higher. As explained in Section 4, large partitions match a disproportionate number of queries.

When necessary, we denote cases where we generate a reduced-size peer descriptor incrementally by inserting the term "incremental" where appropriate. In cases where the peer descriptor is generated with a pre-set K, we insert the term "fixed-K" where appropriate.

The stopping condition of the splitting process is ideally based on a direct measurement of the rate of decrease in Q_d. The problem with using this measure in a stopping condition is that it is not possible to determine the impact of a split on Q_d analytically.

Fortunately, there exists a strong correlation between Q_d and the average N_t^i with increasing K. This correlation, with a correlation coefficient of 0.99954, is shown in Figure 7. This correlation conforms to the fact that there exists a direct relationship between number of terms in a partition and the number of unique queries it matches. Because average N_t^i is measurable during the partitioning process, while Q_d is not, we use the former in formulating our stopping condition.

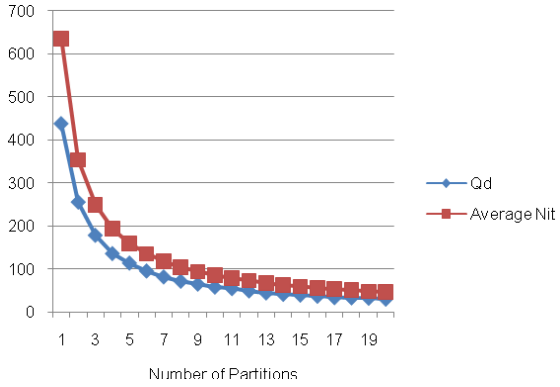


Figure 7. Number of co-occurrence errors and average number of unique terms in each partition with increasing K. In these results, full-sized Bloom filters are generated with fixed-K partitioning.

The stopping condition occurs when a split does not decrease the average N_t^i by threshold percentage U. Because average N_t^i approximates Q_d, the drop in average N_t^i must be large enough to offset the expected increase in Q_c caused by the split. An insufficient decrease in average N_t^i means that the split is unable to reduce Q_f. In the event that the stopping condition is reached, the split is rolled back and the splitting process stops. Formally, the stopping condition is reached if the following is true when trying to create a $(K+1)^{th}$ partition:

$$\frac{avg(N_t^i, K+1)}{avg(N_t^i, K)} - 1 > U$$

In the expression above, avg(N_t^i, K) refers to the average N_t^i given K partitions.

Empirically, we found that U = -0.1 yields the best results, so if a split does not change the average number N_t^i by -10%, we undo the split and stop the splitting process. In Figure 8, we show the performance of incremental partitioning with different U values.

Incremental partitioning with the U = -0.1 stopping threshold results in an average Q_f (304) that is on average within 8% of the average Q_f of the case where we manually tune the optimal K value for each peer (281) – labeled *opt-K* – and 50% lower the Q_f with the base case of K = 1(608).

To show that this improved performance is consistent among all peers in our experiment, we also consider the performance of incremental partitioning on a peer-by-peer basis. In Figure 9, we show the Q_f values for each of the 50 peer collections when: using a single partition (max); when using opt-*K*; and when using incremental partitioning. The consistent closeness of the graphs for opt-*K* and inc indicates clearly the fitness of the incremental partitioning technique and the stopping condition for arbitrary peer collections.

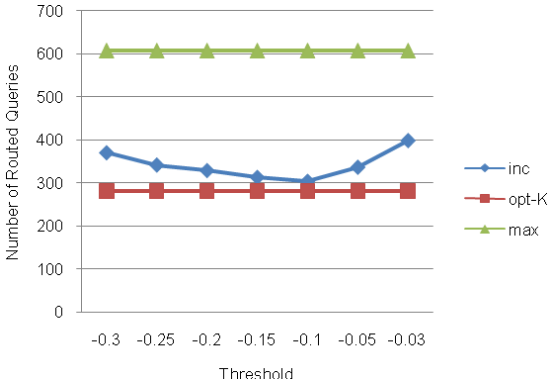


Figure 8. Average number of routed queries with various threshold values. (Note that thresholds should actually be) Max indicates no partitioning.

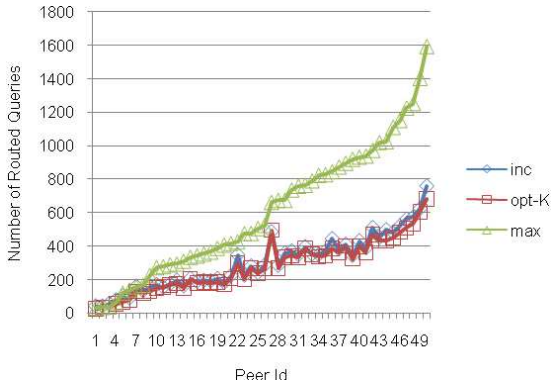


Figure 9. Number of routed queries per peer with incremental partitioning, opt-*K* partitioning and no partitioning (max).

8. OTHER CONSIDERATIONS

8.1 Handling Collection Updates

Over time, users may add or delete files from their collections. These changes must be reflected in the peer descriptions. Handling file addition is straightforward, due to the greedy nature of the ΔM partitioning algorithm. Our experimental results are based on an arbitrary ordering of file files. We expect that additional files will be assigned to likewise appropriate partitions. Updating the corresponding Bloom filters is merely a matter of modifying the appropriate bits that were previously set to 0. Transmitting Bloom filter updates to neighbors using partitioned descriptions may be more efficient, as it is possible to transmit updates to just the modified partitions.

Handling file deletion requires a little more engineering because it requires knowing whether the deletion removed with it the last instance of a particular term from a partition. If so, this term must be removed from the partition's term set and corresponding Bloom filter. This requires maintaining a map between files and partitions as well as maintaining counts of term frequencies in each partition. This is the technique used in other work, such as [12][27][29] and is straightforward to apply to our case.

Under certain conditions, it is also reasonable to handle deletions by ignoring them. In cases where collections are large, for example, it is unlikely that ignoring a single deletion will result in a routing error and therefore ignoring it is worthwhile. Peers that tend to stay online for long periods of time tend to have large collections and delete relatively few files. Peers that join and leave regularly have their descriptions updated when they join. Finally, ignoring deletions does not introduce any false negative routing errors, only, possibly, false positives. For all of these reasons, handling deletions may be relatively unimportant.

Updates to descriptions can be transmitted to neighbors either periodically (e.g., whenever a user logs into the system), when neighbors request updates (as done in Gnutella [3]) or after some change threshold has been reached.

8.1.1 Updates to the incrementally created partitions

One question is whether the partitions need to be updated due to inserts or deletes. These operations affect the average N_t^i, either increasing it (on inserts) or decreasing it (on deletes). If average N_t^i gets too high, then the rate of collisions increases. If average N_t^i gets too low, then the partitions are not aggressive enough in reducing co-occurrence errors.

On an insert, therefore, we attempt to merge two partitions. Merging partitions reduces the collision rate by increasing the number of bits available to encode each Bloom filter.

We attempt to merge the smallest partition (in terms of term set size) with another using ΔM to pick the most appropriate one. We merge the smallest partition in an attempt to keep the partition sizes as even as possible. If the attempted merge does not undo a good split as defined by the stopping condition mentioned in Section 7.3 (that is, allow the merge if $\text{avg}(N_t^i, K + 1) > (1 + U)\ \text{avg}(N_t^i, K)$), then it is allowed. This process repeats until some merge violates the stopping condition for merging. A new Bloom filter is then encoded based on the new partitions.

A similar process occurs when a file is deleted from the collection. A split is attempted on the largest partition (in terms of the term set size). If the split does not violate the stopping condition, then it is allowed. This process repeats until some split violates the stopping condition.

8.2 Handling Multi-Hop Routing

We have so far ignored multi-hop routing in this work because it is ignored in practical P2P file-sharing systems. This is the case because, with flooding, routing cost is concentrated in the last hop. If we flood with a degree f, the ratio of last hop messages to the rest of the messages is $(f - 1) / f$ – very close to 1 for practical values of f, which is on the order of tens.

However, if desired, we provide a rough outline of how to implement multi-hop routing with our technique. One approach is to apply the hierarchical indexing discussed in [24]. With hierarchical routing, each peer transmits to each of its neighbors the summaries of the collections of its other neighbors, which could also contain the summaries of its neighbors' neighbors.

The opportunity that our partitioned collection technique affords is that the index hierarchies can be more precise, leading to greater routing accuracy. Instead of treating each peer as a set of terms, we can treat it as a set of term sets. The specific claim we are making is that our techniques can be applied to hierarchical routing to improve its accuracy. The details of this routing scheme are the subject of ongoing work.

9. CONCLUSION AND FUTURE WORK

Analyses of Gnutella network query logs indicate that over 50% of queries forwarded to a peer return no matching files, wasting both network and computational resources. The problem lies in how peer descriptions are created and used in the query routing process. Each peer is described by its term set (based on file descriptors) and queries are routed to the peer by verifying the existence of query terms in this set. The set of terms, however, may suggest term combinations in shared files that do not actually exist, resulting in incorrect routing decisions.

To increase the resolution of a peer description, we partition its files and create a description for each partition, reducing the number of erroneous term combinations. Experimental results on data from the Gnutella network show that our techniques can reduce the number of incorrectly routed queries by 30% to 60% at virtually no cost. Furthermore, we can incrementally generate an appropriate number of partitions for individual peers based on the distribution of descriptive data in its shared file collection.

Our algorithm for generating partitions is computationally simple and generates solutions that traditional clustering algorithms do not admit. Furthermore, the solution, by its greedy nature, is immediately amenable to file additions while handling file deletions is a matter of simple engineering.

We are currently considering the use of query log data to further increase the accuracy of the partitions. This is important as we do not want to break potential co-occurrences that never actually occur in queries.

Second, we are applying these techniques to improving the routing accuracy of Web corpora. So far, our results are promising.

10. References

[1] C. Rohrs, Query Routing for the Gnutella Network, Limewire LLC Technical Report, http://rfc-gnutella.sourceforge.net/src/qrp.html, December 18, 2001.

[2] B. H. Bloom, Space/Time Trade-offs in Hash Coding with Allowable Errors, ACM CACM, 13(7), July 1970.

[3] T. Klingberg and R. Manfredi, Gnutella Protocol 0.6, Web Document, 2002, rfc-gnutella.sourceforge.net/src/rfc-0_6-draft.html.

[4] B. MacQueen. Some Methods for Classification and Analysis of Multivariate Observations, In *Proc. Berkeley Symp. on Math*. Stat. Prob., 1967.

[5] S. Michel, M. Bender, P. Triantafillou, G. Weikum, IQN Routing: Integrating Quality and Novelty in P2P Querying and Ranking, In *Proc. EDBT*, 2006.

[6] The Partition Problem, Wikipedia.org, 2006.

[7] J. Lin. Divergence measures based on the shannon entropy. IEEE Trans. on Information Theory, 37(1):145—151, January 1991.

[8] J. Lu and J. Callan. User Modeling for Full-text Federated Search in Peer-to-peer Networks. In *Proc. ACM SIGIR*, 2006.

[9] C. Tang, Z. Xu and S. Dwarkadas. Peer-to-Peer Information Retrieval Using Self-Organizing Semantic Overlay Networks. In *Proc. ACM SIGCOMM*, 2003.

[10] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, A. Y. Wu. An Efficient k-Means Clustering Algorithm: Analysis and Implementation. In IEEE Trans. Pattern Anal. And Mach. Learning, 24(7), July, 2002.

[11] I. Podnar, M. Rajman, T. Luu, F. Klemm, K. Aberer Scalable Peer-to-Peer Web Retrieval with Highly Discriminative Keys, In *Proc. IEEE Conf. Data Eng.*, 2007.

[12] G. Koloniari and E. Pitoura, Content-based Routing of Path Queries in Peer-to-Peer Systems, In *Proc. EDBT*, 2004.

[13] A. Frieze, M. Jerrum. Improved Approximation Algorithms for MAX-k-CUT and MAX BISECTION. Algorithmica, 18:61-77, 1997.

[14] I. Giotis, and V. Guruswami, Correlation Clustering with a Fixed Number of Clusters, In *Proc. ACM-SIAM Symposium on Discrete Algorithms*, 2006.

[15] L. T. Nguyen, W. G. Yee, D. Jia, and O. Frieder. A Tool for Information Retrieval research in Peer-to-Peer File Sharing Systems. In *Proc. IEEE ICDE*, Turkey, Apr. 2007.

[16] L. T. Nguyen, D. Jia, W. G. Yee, and O. Frieder. An Analysis of Query Logs in Gnutella Peer-to-Peer Network. In *Proc. ACM SIGIR Conf.*, Amsterdam, July 2007.

[17] I. Stoica, et al. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *Proc. ACM SIGCOMM*, San Diego, Aug. 2001.

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

[19] S. Saroiu, P. K. Gummadi, S. D. Gribble: A Measurement Study of Peer-to-Peer File Sharing Systems. In *Proc. Multimedia Comp. and Networking* (MMCN) 2002.

[20] J. Callan, Z. Lu, and B. Croft. Searching Distributed Collections with Inference Networks. In *Proc. ACM SIGIR*, 1995.

[21] L.Gravano, C. Chang, and H. Garcia-Molia. GlOSS: Text-Source Discovery over the Internet. ACM Trans. Database Sys. 24(2), 1999.

[22] L. Si, R. Jin, J. Callan, and P. Ogilvie. A Language Modeling Framework for Resource Selection and Results Merging. In *Proc. ACM CIKM*, 2002.

[23] M. Shokouhi, M. Baille, and L. Azzopardi. Updating Collection Representations for Federated Search. In *Proc. ACM SIGIR*, 2007.

[24] A. Crespo, H. Garcia-Molina. Routing Indices for Peer-to-Peer Systems. In *Proc. IEEE ICDCS*, 2002.

[25] B. Yang and H. Garcia-Molina. Efficient search in peer-to-peer networks. In *Proc. IEEE ICDCS*, 2002.

[26] Q. Wang, M. T. Ozsu, An Efficient Eigenvalue-based P2P XML Routing Framework. In *Proc. IEEE P2P*, 2007.

[27] L. Fan, P. Cao, J. Almeida, A. Z. Broder. Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol. IEEE/ACM Trans. Networking, 8(3), June, 2000. pp. 281-293.

[28] W. G. Yee, L. T. Nguyen, O. Frieder, A View of the Data on P2P File-sharing Systems. In *Proc. Wkshp. Large Scale Dist. Inf. Sys. Inf. Retr.*, 2007.

[29] D. Guo, J. Wu, H. Chen, X. Luo. The Dynamic Bloom Filters. In *Proc. IEEE Infocom*. 2006.

[30] M. Shokouhi, M. Baillie and L. Azzopardi. Updating Collection Representations for Federated Search. In *Proc. ACM SIGIR*. 2007.

[31] L. R. Monnerat and Cláudio L. Amorim. D1HT: a distributed one hop hash table. In *Proc. IEEE IPDPS*, 2006