Automatic Tuning of File Descriptors in P2P File-Sharing Systems

Dongmei Jia, Wai Gen Yee, Ophir Frieder
Illinois Institute of Technology
Chicago, IL 60616, USA
jiadong@iit.edu, yee@iit.edu, ophir@ir.iit.edu

ABSTRACT

Peer-to-peer file-sharing systems have poor search performance for rare or poorly described files. These files lack a quality or variety of metadata making them hard to match with queries. A server can alleviate this problem by gathering the descriptors used by peers via what we call probe queries, and use these descriptors to improve its own. We consider probe query triggering mechanisms and criteria for selecting a file for which to probe in this work. Experimental results indicate that probe queries are effective in improving search performance.

Categories and Subject Descriptors

H.3.5 [**Information Storage and Retrieval**]: Online Information Services – *Web-based services.*

General Terms

Algorithms, Performance, Experimentation.

Keywords

Metadata management, P2P Search.

1. INTRODUCTION

Peer-to-peer (P2P) file-sharing systems like Limewire's Gnutella [3] are extremely popular with millions of daily users sharing petabytes of data [9]. These systems are highly effective in locating popular files, but less so for rare and poorly described ones [1]. [2] shows that different ranking functions may improve the identification of rare files in query result sets, but does not address the issue of retrieving these files from servers in the first place. Only after a result is returned to a client can it be ranked. Hence, to improve a query's performance, a good approach is to improve its recall (i.e., its likelihood of matching relevant files).

Improving recall is an important goal of P2P file-sharing systems. Ostensibly, peers publish files because they desire to share them. For example, a new music group may want to publicize its new song or a P2P client may want others to download its files to improve its reputation in the system.

The problem with search in P2P file-sharing systems is that, although multiple replicas of a file may exist, they are maintained at different servers, and are described with small, user-defined descriptors, which are usually filenames. Therefore, although the system's aggregate metadata for a file might effectively describe it, it is possible that no single descriptor is sufficient.

Ninth International Workshop on the Web and Databases (WebDB 2006), June 30th, 2006, Chicago, IL, USA.

This negative consequence of the distributed nature of the description of files in P2P file-sharing systems manifests itself in poor search performance. In practical P2P file-sharing systems, queries are conjunctive: a file matches a query if all query terms are contained in the file's descriptor [6]. Consider an example where D1, D2 are two descriptors of replicas of the same file, F, from two different servers, and Q is a query. If D1 = {t1}, D2 = {t2}, and Q = {t1, t2}, where t1 and t2 are terms, the evidence is strong that F is relevant to Q. However, because of the (conjunctive) matching criterion, neither D1 nor D2 are returned as Q's result.

Our work focuses on increasing the richness of the descriptors of shared files by the use of *probe queries*, which we define as a query meant to search the network for other replicas' descriptors of a given file. With the results of these queries, the client can discover alternate ways of describing its local replica of the file. For example, the server of D1 in the example above may send a probe for F, yielding D2. The server can then transform D1 into D1' = $D1 \cup D2$ = {t1, t2}. With D1', the server can correctly match its replica of F with Q.

The basic goal of a probe query for a file is to automatically aggregate the available metadata associated with a replica of the file that exist on other peers in the system, and then incorporate these metadata into the local replica's descriptor. The enhanced local descriptor makes the replica more identifiable, increasing the likelihood that a query correctly matches it.

In this paper, we consider the design of a probing system in a P2P file-sharing system. We cover the issues of determining when a peer should probe, what file it should probe, and what it should do with the probe results.

2. RELATED WORK

Recent work proposes to identify rare files in search results [1][2]. However, these works do not consider the more basic problem of retrieving the rare data in the first place.

Limewire's Gnutella attempts to improve recall and limit bandwidth consumption by increasing the time-to-live of queries that yield few results [3]. Again, this system assumes that data are described adequately enough to be returned by a query.

There are also classes of work that strive to improve search in P2P environments that share text documents [4]. Such systems do not suffer from query over-specification because the sought-after document is self-describing, and does not rely on small, user-defined descriptors for matching.

Other work tries to improve performance by "intelligently" routing queries to the most-relevant servers [5]. These works assume that files are adequately described for queries to return matches.

Some work assumes that file descriptors are structured [17]. Structure constrains queries and may improve performance. Although our application assumes that descriptors are small and unstructured, there is no reason why our techniques cannot be applied to the structured data environment.

Our approach bears some resemblance to work that either adds terms to queries or to descriptors to improve recall [16]. However, in practice, there exists no standard mechanism to for adding terms and it is unclear what impact this would have on query performance or cost. Moreover, such techniques are generally not applicable when queries are conjunctive.

In general, our work takes a different approach, focusing on the client application level by improving recall by enhancing data description.

3. MODEL

Peers collectively share (or publish) a set of (binary) files by maintaining local replicas of them. Each replica is represented by a descriptor, which also contains an identifying key (e.g., an MD5 hash on file's bits). All replicas of the same file naturally share the same key. A query issued by a client is routed to all reachable peers until the query's time-to-live expires. The query matches a replica if it is contained in the replica's descriptor. For each match, the server returns its system identifier and the matching replica's descriptor.

Formally, let O be the set of files, M be the set of terms, and P be the set of peers. Each file $o1,o2\in O$ has an identifier associated with it, denoted k_{o1}, such that $k_{o1}=k_{o2}$ if and only if o1=o2. We also refer to k_{o1} as the *key* of file o1 (e.g., the MD5 hash value mentioned above).

Each file $o\in O$ has a set of terms that *validly* describe it. We denote the set of valid terms for o as $T_o\subseteq M$. Intuitively, T_o is the set of terms an average person would "most likely" use to describe o. Each term $t\in T_o$ has a strength of association with o, denoted soa(t, o), where $0\leq \mathrm{soa}(t, o)\leq 1$ and $\sum_{t\in T_o}\mathrm{soa}(t, o)=1$. The strength of association a term t has with a file o describes the relative likelihood that it is to be used to describe o, assuming all terms are independent. The distribution of soa values for a file o is called the *natural term distribution* of o.

A peer $s\in P$ is defined as a pair, (R_s, g^s), where R_s is the peer's set of replicas and g^s is the peer's unique identifier. Each replica $r^o_s\in R_s$ is a copy of file $o\in O$, maintained by peer s, and has an associated descriptor, $d(r^o_s)\subseteq M$, which is a multiset of terms that is maintained independently by s. Each descriptor $d(r^o_s)$ also contains k_o, the key of file o. The number of terms that a descriptor can contain is fixed.

A query $Q^o\subseteq T_o$ for file o is also a multiset of terms. The terms in Q^o are expected to follow o's natural term distribution. When a query Q arrives at a server s, the server returns *result set* $U^Q_s=\{(d(r^o_s), g^s) \mid r^o_s\in R_s \text{ and } Q\subseteq d(r^o_s) \text{ and } Q\neq\emptyset\}$—membership in the result set requires that a result's descriptor contain all query terms, in accordance with the matching criterion,.

The client receives result set $U^Q=\cup_s U^Q_s$, $s\in P$, and groups individual results by key, forming $G=\{G_1, G_2, \ldots\}$, where $G_i=(d_i, i, l_i)$, $d_i=\{\oplus d(r^i_s) \mid (d(r^i_s), g^s)\in U^Q \text{ and } k_i=i\}$ is the group's descriptor, i is the key of G_i, and $l_i=\{g^s \mid (d(r^i_s), g^s)\in U^Q \text{ and } k^i=i\}$ is the list of servers that returned the results in G_i. In this definition, $\oplus$ is the multiset sum operation.

The client assigns a rank score to each group with function $F_i\in F$, defined as $F: 2^M\times 2^M\times\mathbb{Z}\times\mathbb{Z}\rightarrow\mathcal{R}^+$. If $F_i(d_j, Q, |G_j|, time_j) > F_i(d_k, Q, |G_k|, time_k)$, where G_j, G_k are groups, then we say that G_j is ranked higher than G_k with respect to query Q. In these definitions of F, $|G_j|$ is the number of results contained in a group, and $time_j$ is the creation time of the G_j (i.e., the time when the first result in G_j arrived).

In commercial systems, such as various versions of Gnutella and eDonkey, popular P2P file-sharing systems, the ranking function is based on group size:

$$F_G(d, Q, a, b) = a.$$

Descriptors in these systems are generally implemented via filenames, although a small amount of descriptive information may be embedded in the actual replica (e.g., ID3 data embedded in MP3 files [14]).

Result keys are generated by MD5 or SHA-1 hashing, and results are grouped based on these keys. Furthermore, when a client downloads a file, the descriptor of this new replica is initialized as a duplicate of one of the servers in the result set.

To simplify our explication, we will use the term "result" informally to describe either a group or an individual result, and clarify the usage if necessary. We refer to the collective set of terms contained in descriptors as metadata.

4. PROBING

4.1 Implementation of Probe Queries

A probe query for a file o is implemented as a query that contains a single term – the key of a file, k_o. Because, by assumption, every descriptor contains a key, this query is guaranteed to match all the replicas of a particular file that it encounters. The use of keys to find replicas occurs in practice. The popular P2P file-sharing system, DC++ (dcpp.net), for example, relies on the existence of keys to identify other sources of a file for multi-source downloading.

Probe query routing can be done via the same routing mechanism used by ordinary queries – controlled flooding via "ultrapeers" [7]. Alternatively, due to the simple nature of the query, an inverse index, consisting of a key and a list of servers can be created on top of a DHT-like routing infrastructure or other indexing mechanisms (e.g., [15]) to save network resources [8].

4.2 Steps in Probing

There are four steps to probing: First, some mechanism triggers the probe in the client. If so, then the client selects a file to probe, collects the results, and then applies the results to the descriptor of the probed file.

4.2.1 Triggering the Probe

In the base case, peers can probe at regular or randomly distributed intervals. However, more selective probing makes better use of system resources. For example, a peer that is relatively busy should not bother probing, as it is already burdened. Rather, a peer that is unsuccessfully trying to share its large library should probe. By doing so, it also relieves the burden of the other peers.

We identify three factors that help us to determine the appropriateness of probing for a peer: number of queries received (N_q), number of responses returned (N_r), and number of files published (N_f). Given a user-defined threshold T, if the following condition holds, the peer performs a probe:

$$T < N_f N_q / (N_r+1) - N_p T,$$

where N_p is the number of probes the peer has already performed.

The probe threshold condition makes N_r the basic metric that describes a peer's participation or utilization in the system. A high N_r can be taken to mean two things: either the peer is adequately advertising its shared files, and/or it is busy. In either case, the peer should not increase its workload by probing.

The degree of participation, however, should be in proportion to the desire of the user to participate, which we measure by N_f. Therefore, a peer that shares many files (high N_f) but does not respond to many queries (low N_r) should probe.

The motive force behind probing, however, is the total activity level in the system, as perceived by the client. If there is a low level of activity in the system, measured by N_q, then probing is a waste of resources. Only in an active system should probes be performed. We therefore add N_q to the threshold condition.

The given condition, therefore, measures the number of responses per shared file and per query. By fixing T over all peers, we tacitly assume that all files are equally popular.

The second term in the condition above "resets" the threshold condition after each probe.

4.2.2 *Selecting a File to Probe*

Once a peer has decided to perform a probe, it must identify a file to probe. Because our goal is to utilize all peers, the logical choice is the file that most contributed to satisfying the probe threshold. By definition, this is the file that has been returned the fewest times as a query result. Let N^i_m be the number of times a local replica r^i has matched a query (i.e., the number of times it has been returned to a client). A file-picking criterion, therefore, is

Criterion 1: Probe r^i, where N^i_m is minimum over all i.

In other words, a low N^i_m indicates that r^i is not being returned at an adequate rate, and may need help in its description.

If we assume that N^i_m is a measure of file popularity, one of the features of Criterion 1 is that it allows the less popular data items to be probed. The benefit of this is that probing popular files should not be necessary as the likelihood that a query does not match at least one of the available replicas is slim.

Another option is to select a file based on how it is described. A file with a small descriptor is not likely to be found because queries over-specify them (i.e., contains terms that are not in the file's descriptor). Let $|d(r^i)|$ be the number of unique terms in the descriptor $d(r^i)$ of r^i.

Criterion 2: Probe r^i, where $|D_i|$ is minimum over all i.

Both of the file-picking criteria we just described suffer from the inability to handle situations where probing has no effect on either N^i_m or $d(r^i)$. In such a case, the same file will be probed repeatedly to no use. To solve this problem, we use Criterion 1 as our file-picking criterion, and after every probe of r^i, we do the following:

$$\text{If } N^i_m = 0 \text{ then } N^i_m \leftarrow 1 \text{ else } N^i_m \leftarrow 2\, N^i_m.$$

By doubling the metric used in Criterion 1, we reduce the likelihood of all probes being performed by a particular peer are for the same file. Such a situation may still arise, of course, but only in degenerate cases where a particular file is very unpopular. In the event of a tie in Criterion 1, we use Criterion 2.

4.2.3 *Copying Terms to the Local Descriptor*

Once probe results for r^i are returned to the client, they are grouped into a multiset. The client then selects terms from this multiset to add to d(ri), the local descriptor of r^i.

As discussed in [2], there are many ways to copy terms into the local descriptor: randomly, by frequency, and so on. To simplify the presentation of results, we randomly copy terms from the probe results into $d(r^i)$, biasing the likelihood that a term is copied by its relative frequency. For example, a term that occurs twice as frequently as another in the probe results is twice as likely to be copied. This process repeats until the local descriptor is full.

In our experiments, copying the most frequent terms also does well, but at the expense of slightly higher cost. We therefore leave out these results.

5. EXPERIMENTIAL RESULTS

We simulate the performance of a P2P file-sharing system to test the large scale performance of our methods. In accordance with the model described in [10] and observations presented in [11], we enhance our experimental model with interest categories, which model the fact that some users have stronger interests in some subsets of data than other. We partition the set of files, O, into sets C_i, where $C_i \subseteq O$, $C_i \cap C_j = \emptyset$ if $i \neq j$, and $\cup_i C_i = O$. At initialization, each peer $s \in P$ is assigned some interests $I_s \subseteq \cup_i C_i$, and is allocated a set of replicas R_s from this interest set: $R_s = \{r^o_s \mid o \in \cup_i C_i$, where $C_i \in I_s\}$. For each replica, r^o_s, allocated at initialization, $d(r^i_s) \subseteq T_i$, where term allocation is governed by natural term distributions. Peer s's interest categories also constrain its searches; it only searches for files from $\cup_i C_i$, where $C_i \in I_s$.

Each category C_i has an assigned popularity, b_i, which describes how likely it is to be assigned to a peer. The values of b_i follow the Zipf distribution [10].

Within each interest category, each file varies in popularity, which is also skewed according to the Zipf distribution [10]. This popularity governs the likelihood that a peer who has the file's interest category is either initialized with a replica of the file or decides to search for it.

Peers in our simulator are populated with TREC data from the 2GB Web track (WT2G), where Web domains, documents in the domains, and terms in the documents are mapped to interest categories, files in categories, and files' valid terms, respectively. Natural term distributions are based on the term distributions within the Web pages. An initial set of replicas with random descriptors of associated terms is allocated to peers based on pre-assigned interest categories. Queries for files are generated using valid terms with a length distribution typical of that found in Web search engines [12] as shown in Table 1 and also exhibited in our P2P file-sharing system query logs.

The simulation parameters shown in Table 2 are based on observations of real-world P2P file-sharing systems and are comparable to the parameters used in the literature.

The data set used consists of an arbitrary set of 1,000 Web documents from 37 Web domains. Terms are stemmed, and HTML markup and stop words are removed. The final data set contains 800,000 terms, 37,000 of which are unique.

We also conducted experiments on other data sets with other data distributions, but, due to space constraints, we only present a representative subset of our results. The data we used for these experiments can be found on our Web site [13].

Table 1-Query Length Distribution.

Length	1	2	3	4	5	6	7	8
Prob.	.28	.30	.18	.13	.05	.03	.02	.01

Although other behavior is possible, we assume that the user identifies and downloads the desired result group with a probability 1/r, where r≥1 is its position in the ranked set of results.

Table 2-Parameters Used in the Simulation.

Parameter	Value(s)
Num. Peers	1000
Num. Queries	10,000
Max. descriptor size (terms)	20
Num. terms in initial descriptors	3-10
Num. categories of interest per peer	2-5
Num. files per peer at initialization	10-30
Num. trials per experiment	10

Performance is measured using a standard metric known as mean reciprocal rank score (MRR) [18], defined as

$$MRR = \frac{\sum_{i=1}^{N_q} \frac{1}{rank_i}}{N_q},$$

where N_q is the number of queries and $rank_i$ is the rank of the desired file in query i's result set. MRR is an appropriate metric in applications where the user is looking for a single, particular result.

For reference, we also present precision and recall, which have slightly different definitions than they do in traditional IR, due to the fact that replicas exist in the P2P file-sharing environment, and assuming that queries are for particular files. Let A be the set of replicas of the desired file, and R be the result set of the query. Precision and recall are defined as:

$$precision = \frac{|A \cap R|}{|R|},$$

$$recall = \frac{|A \cap R|}{|A|}.$$

These more traditional IR metrics are useful in roughly diagnosing the performance of query processing and in generalizing the presented performance to other domains.

5.1 Triggering the Probe

We now consider the impact of various probe triggering techniques on performance. The three techniques we consider are:

1. No probing.
2. Using the threshold, described in Section 4.2.1.
3. Randomly selecting a peer to probe.

For the probing cases, we perform 5,000 probes. To do this using the probe threshold, we tune T so that, after 10,000 queries, approximately 5,000 probes are issued. For random probing, we assign to each peer a probability of probing during each iteration of the simulation that results in 5,000 probes after 10,000 queries.

Experimental results shown in Figure 1 clearly indicate that probing improves query performance. Probing randomly increases MRR by 20%. Probing using the threshold, however, increases MRR by 30%.

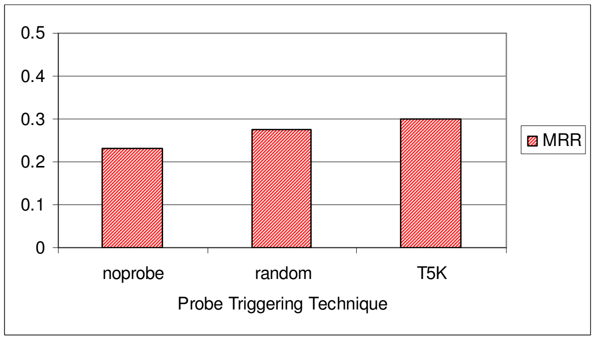

Figure 1-MRR with Various Probe Triggering Techniques.

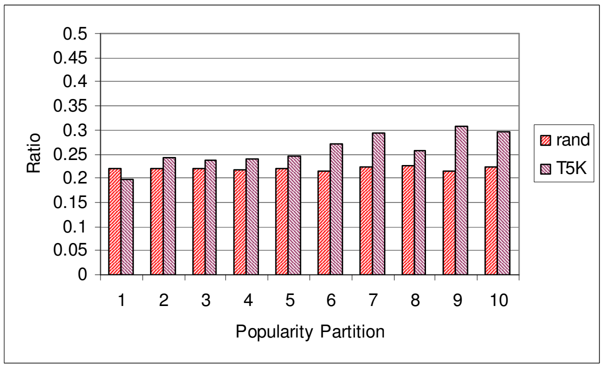

Figure 2-Ratio of Number of Probe Queries to Number of Files for Various Popularity Partitions.

The reason for this performance improvement is because probes are correctly being performed by under-utilized peers. This is suggested by the fact that probes are generally directed toward the files that are perhaps more difficult to find due to their lower popularity. To verify this, we partitioned the set of files into ten *popularity partitions*. Each partition represents 10% of the files, where partition 1 contains the most popular 10% of files, partition 2 contains the next 10% most popular files, and so on. In Figure 2, we show the ratios of number of probe queries issued for files in each popularity partition and the number of replicas in each popularity partition. The results indicate that more of the probes are being performed on less popular data when using the threshold to pick files. As expected, when probing is performed randomly, the ratios remain constant over all partitions, as all replicas are equally likely to be probed.

We also ran experiments where we varied the threshold T. To control these experiments, we set T to values such that, after 10,000 queries, there were 2,500, 5,000, 7,500, and 10,000 probes. The results, shown in Figure 3 are intuitive. As the number of probes increases, so does performance. However, the rate of performance increase decreases with an increasing number of probes. Additional probing after most files have been probed has marginal value.

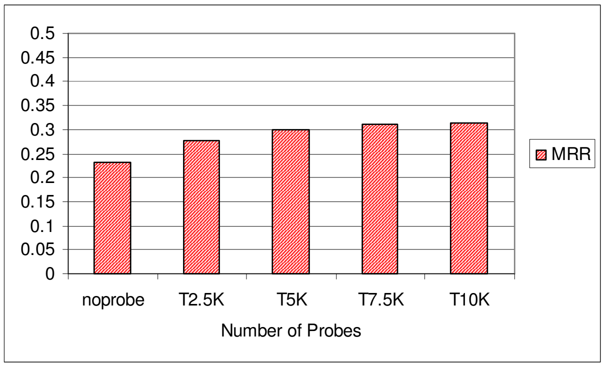

Figure 3-Effect of Various Probing Rates on MRR.

5.2 Alternative Probe File Selection Techniques

We now consider the performances of two alternative file selection techniques:

1. Randomly selecting a file to probe.
2. Selecting a file to probe based on the criteria discussed in Section 4.2.2.

These experiments assume that we are triggering probes using the threshold T5K (T tuned to perform 5,000 probes after 10,000 queries).

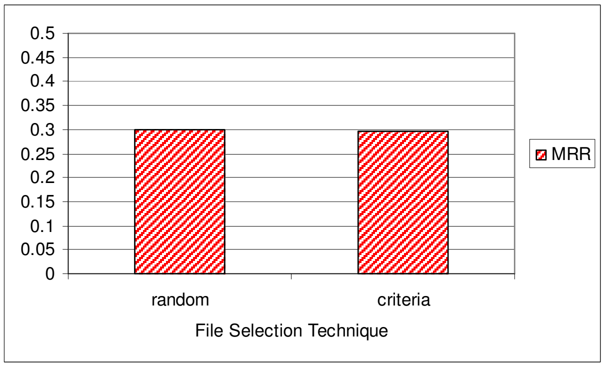

Figure 4-MRR with Various Probe File Selection Techniques.

The results of these experiments, shown in Figure 4, suggest that the results are a wash. There is very little difference in MRR between the two techniques. A deeper analysis, (not shown here), shows that a larger proportion of probes, when using the "criteria," are directed toward already popular files. This is wasteful in two ways. First, the probes are unnecessary for popular files, and, second, because popular files are queried frequently, cost, in terms of the number of results per query, (which we consider later), increases unnecessarily.

In other words, there is significance to the different file selection techniques that are not revealed by MRR in these experiments. We have used other file selection techniques that maximize the identification of rare files, but often at the expense of overall MRR. More in-depth treatment of other file-selection techniques is the subject of ongoing work.

5.3 Cost Analysis

We define cost as the number of query responses received by the client. This metric roughly estimates the amount of work the client must perform to process a query. More importantly, this metric roughly estimates network cost in a topology-independent way.

Probing increases the cost of each query. By enhancing data description, we also increase the likelihood that a query will match some file. The increase in cost can be significant.

To counter this cost, we propose server-side Bernoulli sampling of the result set for each query. That is, for each matching result for a query, the server decides to return it to the client with a fixed probability Pr, $0 \leq Pr \leq 1$. This type of sampling is expected to preserve the overall distribution of terms and results in the result set. It also allows us to predictably reduce the cost by a factor Pr. In the experiments in this section, we arbitrarily use "criteria" probe file selection. The results for random probe file selection are similar.

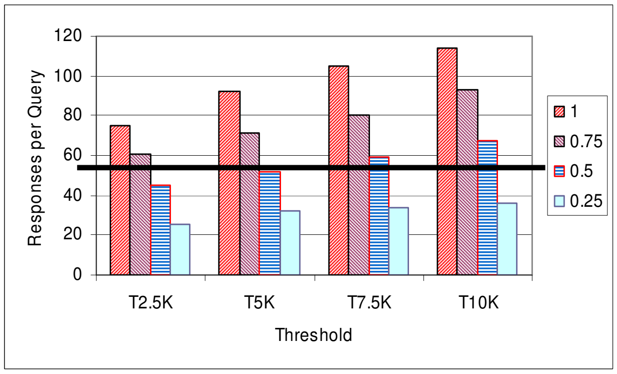

Figure 5-Responses per Query for Different Probing and Sampling Rates. The black line indicates cost without probing.

As shown in Figure 5, cost increases with probing range from 36% to over 100% with varying thresholds if sampling is not used. Predictably, with sampling, costs are reduced by, approximately, a factor Pr. The cost decrease factor is slightly greater than Pr because, in a well-running P2P file-sharing system, the average number of results per query is high because more peers are actively sharing more files.

Sampling, in fact, is able to reduce the cost of probing to levels below that of not probing with no sampling. This decrease in cost from the base case can be over 50%.

Sampling, however, has a negative impact on MRR. This is the case because it is likely that, for some queries, the desired result will be selected out of the result set. The question is whether the decrease in MRR offsets the improvements in cost.

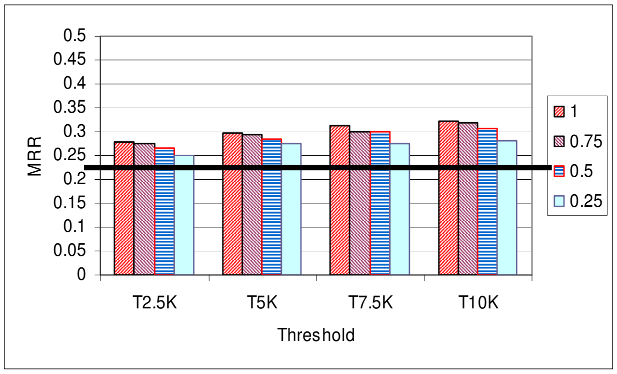

Figure 6-MRR with Different Probing and Sampling Rates. The black line indicates MRR without probing.

Fortunately, MRR decreases at a slower rate than cost, as shown in Figure 6. MRR decreases by at most about 15% with a sampling rate of 25%. However, in these experiments, MRR is never worse than when not using probing. For example, when using T10K probing and 25% sampling, MRR is approximately 20% better than when not probing, and cost is 35% lower. Probing with sampling can therefore lead to a win-win situation in terms of both ranking performance and cost. Based on this performance, it seems likely that a good way of designing a probing system would be to maximize probing rate, and then reduce cost, via sampling, as necessary.

The reason for this positive performance/cost behavior is due to the effect of probing on recall and precision. Result sets are generally of a higher quality in terms of these two metrics as shown in Figure 7. The increased precision in particular, reduces the likelihood that sampling will eliminate all relevant results from a result set.

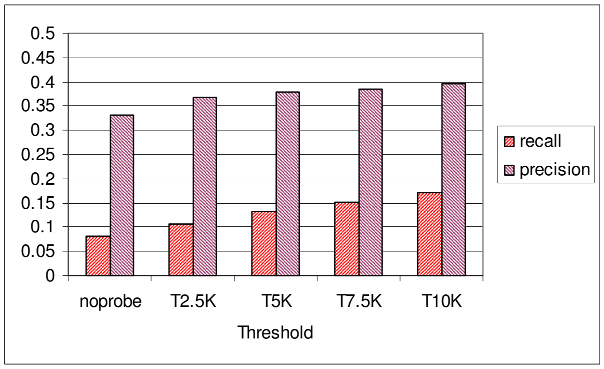

Figure 7-Recall and Precision with Various Probing Rates.

6. CONCLUSION

Given the conjunctive matching criterion of today's P2P file-sharing systems, poor data description limits overall performance. Probe queries help solve this problem by automatically tuning local descriptors using those of peers. Our experimental findings demonstrate that it is possible to improve performance with probes with very little (potentially negative) cost.

We are now considering ways of better controlling exactly where probes are directed (i.e., more or less popular files). We are also working on models that help in tuning threshold and sampling values in a distributed manner.

7. REFERENCES

[1] B. Loo, J Hellerstein, R Huebsch, S Shenker and I Stoica. Enhancing P2P File-sharing with an Internet-Scale Query Processor. In *Proc,VLDB Conf.* August, 2004.

[2] W. G. Yee, D. Jia, and O. Frieder, Finding Rare Data Objects in P2P File-sharing Systems, *In Proc. of the Fifth IEEE Intl. Conf. on Peer-to-Peer Computing*, Germany, September 2005.

[3] Limewire. http://www.limewire.org.

[4] J. Lu and J. Callan, Federated Search of Text-Based Digital Libraries in Hierarchical Peer-to-Peer Networks, *Proc. Euro. Conf. on Inf. Retr.*, 2005.

[5] K. Sripanidkulchai and B. Maggs and H. Zhang, Efficient Content Location Using Interest-Based Locality in Peer-to-Peer Systems, *Proc. IEEE INFOCOM*, 2003.

[6] C. Rohrs, Keyword Matching [in Gnutella], *LimeWire Technical Report*, Dec., 2000, www.limewire.org/techdocs/KeywordMatching.htm.

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

[8] I. Stoica, Robert Morris, D. Karger, F. Kaashoek and H. Balakrishnan, Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications, *Proc. SIGCOMM*, 2001.

[9] Slyck.com P2P File-sharing Statistics. http://slyck.com/stats.php

[10] M. T. Schlosser, T. E. Condie, and S. D. Kamvar. Simulating a file-sharing p2p network. *In Proc. Wkshp. Semantics in Peer-to-Peer and Grid Comp.*, May 2003.

[11] S. Saroiu, P. K. Gummadi, and S. D. Gribble. A measurement study of peer-to-peer file sharing systems. *In Proc. Multimedia Computing and Networking* (MMCN), Jan. 2002.

[12] P. Reynolds and A. Vahdat. Efficient peer-to-peer keyword searching. In *Proc. ACM Conf. Middleware*, 2003.

[13] PIRS Research Group Data, http://ir.iit.edu/~waigen/proj/pirs/data/

[14] M. Nilsson. Id3v2 web site. Web Document, 2006. www.id3.org.

[15] H. V. Jagadish, B. C. Ooi, K.-L. Tan, Q. H. Vu, and R. Zhang. Speeding up Search in Peer-to-Peer Networks with A Multi-way Tree Structure. In *Proc. SIGMOD*, 2006.

[16] Md. Mehedi Masud, Iluju Kiringa, Anastasios Kementsietsidis. Don't Mind Your Vocabulary: Data Sharing Across Heterogeneous Peers. In *Proc. of the Intl. Conf. on Coop. Inf. Sys.* (CoopIS), 2005.

[17] S. Abiteboul, I. Manolescu and E. Taropa, A Framework for Distributed XML Data Management. In *Proc. EDBT*, 2006.

[18] E. Voorhees and D. Tice, "The TREC-8 Question Answering Track Evaluation," In *Proc. of the Eighth Text REtrieval Conference*, 1999.