On Search in Peer-to-Peer File Sharing Systems

Wai Gen Yee
Information Retrieval Lab
Illinois Institute of Technology
10 West 31st Street
Chicago, IL 60616 USA
yee@iit.edu

Ophir Frieder
Information Retrieval Lab
Illinois Institute of Technology
10 West 31st Street
Chicago, IL 60616 USA
ophir@ir.iit.edu

ABSTRACT

We consider the problem of information retrieval in a peer-to-peer file sharing system. We assume that peers are unreliable, metadata are sparse, and queries are short. In light of this, the question becomes whether there exists a combination of metadata management and ranking techniques that can consistently improve query results. We try several alternatives, and, through analysis and simulation, show a combination that generally yields the best results.

1. INTRODUCTION

Peer-to-peer (P2P) file sharing is one of the leading applications of P2P technology. In such a system, there is no centralized authority, so users are free to join and leave at any time. One of the problems of such a system, however, is that it may be hard to find desired data. We consider the problems of using traditional information retrieval (IR) techniques in a P2P environment, and determine ways by which users can best find desired data.

In particular, a P2P file sharing environment has the following characteristics:

1. Short queries - In our application domain, users generally find data objects (files) by comparing them to queries, not by comparing them to other data objects. Queries are manually created, and are therefore generally short.
2. Sparse metadata sets - Shared data objects are often binary files that are generally annotated by retrieving data from simple Web databases [4] or by hand. The available metadata are generally sparse.
3. High **churn** rates - Churn refers to tendency for peers to join and leave the network. Researchers have observed a high churn rate in P2P file sharing systems [13].

Considering these characteristics, many problems arise that are not common in traditional IR. Because metadata are sparse and queries are short, it may be difficult to distinguish and rank data objects. Because of the high churn rate, the set of shared data is constantly changing, and it is therefore difficult to maintain the accurate statistics required by many IR techniques. For example, Google uses statistics on the topology of the Web to rank pages. The Web's topology is relatively static, so these statistics can be assumed to be valid until the next Web *crawl*.

Our P2P IR system performs three major functions:

1. P2P file sharing - Each peer can serve and request files, route queries, and handle peers joining and leaving the network;
2. Metadata management - One of the features of a P2P IR system is that a peer has the ability to function as a server for data objects. Since a data object is identified by metadata, and each client manages its own metadata, the client can control the effectiveness of its service by judiciously managing its metadata;
3. Result ranking - Given that a query may return many ambiguous results, it is important to correctly rank them. Good ranking reduces the cognitive load on the user and computational load on the client.

Commonly available P2P file sharing systems (e.g., LimeWire's Gnutella implementation) either have or can be extended to have these three parts. The question we address is, considering our application domain, how should the second and third parts, which are essential to IR, be designed?

2. RELATED WORK

Much of today's work in P2P IR focuses on identifying highly reliable peers, and giving them specialized roles in statistics maintenance, indexing, and routing [7, 8, 17]. The performance of such systems is impressive; however, the application domain is different than the one we consider. We make no assumptions about the relative capabilities of the peers, and our work is therefore more applicable to ad hoc environments, where functionality is fully distributed among all participants. Put another way, our work focuses on query management from the perspective of clients, whereas previous work focuses on query management from the perspective of servers.

Our work also bears many similarities to that of metasearch engines [9]. The problems related to such systems include source selection and merging of results from independent sources. Again, metasearch engines operate in a highly stable environment, and solutions such as sampling sources to

SAC'05 March 13-17, 2005, Santa Fe, New Mexico, USA

determine content and using ontologies for ranking results have been developed [3].

There are also systems that discriminate among peers based on their past behavior. Results coming from certain peers' may be ranked higher based on previous responses to queries [16]. These systems require that statistics be maintained on peers, which may not be scalable in a large system. This technique may lead to the formation of *communities* within the network as well. Communities are useful when members have similar interests. Communities, however, may make the search for unexpected data objects harder and may make the network more brittle.

Finally, our approach is conceptually different from previous approaches in that we consider peers to have some influence in the form of the body of metadata for the shared data objects in a P2P system. This control implies an influence (albeit small) in the way data objects are described and thereby ranked. We also leave it up to the client to rank query results. Of course, extensions to our system, like the ones described in the cited work are possible, but ultimately in a P2P environment, there must be some guessing as to the current state of the data in the system.

3. MODEL

In our model, we assume that the peers of a P2P system collectively **share** a set of **data objects**. Each data object is a file (e.g., a music file), which is wholly identified by a **descriptor**. A descriptor is a **metadata set**, which is composed of **terms**. Depending on the implementation, a term may be a single word or a phrase. (A metadata set is technically a *bag* of terms, because each term may occur multiple times.) We assume that a data object is a binary file. This generally precludes directly searching the data object.

A peer acts as a **client** by initiating a **query** for a *particular* data object (as opposed to any one of a *category* of data objects). A query is also a metadata set, composed of terms that a user thinks best describe the desired data object. A query is routed to all reachable peers, which act as **servers**. Query **results** are references to data objects that fulfill the **matching criterion**:

$$D_O \supseteq Q, \tag{1}$$

where D_O is the descriptor of data object O, and Q is the query. In other words, the data object's descriptor must contain all the query terms.

A query result contains the data object's descriptor as well as the identity of the source server. The descriptor helps the user distinguish the relevance of the data object to the query, and the server identity is required to initiate the data object's download.

Once the user selects a result, a local replica of the data object is made. In addition, the user has the option of manipulating the replica's descriptor. She may manipulate it for personal identification or to better share it in the P2P system.

Our model is independent of the underlying routing protocol. This is a deliberate design decision that maximizes its generality. As a basis, we assume Gnutella-style flooding [6], but our model does not preclude more advanced routing protocols, such as shortcuts [16] or those allowed by hybrid networks [15].

4. IR COMPONENTS IN OUR P2P SYSTEM

Our P2P IR system is composed of metadata management and IR extensions built on top of a P2P file sharing system. It is designed to be compatible with existing systems, be computationally simple, and make as few architectural assumptions as possible. These constraints together maximize its usability and likelihood of adoption. The question is whether our IR extensions can offer any benefits while satisfying these constraints. Before answering this question, we first describe the general operation of our P2P IR system. We then describe its two major components: the metadata management system and the ranking system.

4.1 Overview of our P2P IR System

We assume that each instance of a (binary) data object has a hash key (e.g., SHA-1 [10]) computed for it using a well-known hash function. Therefore, all instances of a given data object have the same hash key. The use of hash key for shared data is a common practice [2], typically for validation and identification purposes. This hash key is inserted into the instance's descriptor as a unit of metadata.

When a client issues a query, results are *grouped* based on hash key. Each group consists of a descriptor, which contains the collective metadata of all contained results, and pointers to their servers. Groups are ranked based on the contents of their descriptors using the ranking system.

When a group is selected, an instance of the corresponding data object is created on the client. The client initializes a descriptor for this data object with the data object's hash key. It then adds additional term from the group's descriptor using the metadata management system. When this is done, the client becomes a server for this data object. We now discuss the metadata management and ranking systems in more detail.

4.2 Metadata Management

The goal of the metadata management system is to build a **body of metadata** in the *spirit of P2P* that best supports high quality query results: many users independently do some metadata management work in a way that improves overall query performance. Because metadata analysis is the means by which data objects are identified, the proper management of metadata is critical.

Today's P2P file sharing systems manage metadata in a way that increases the *reliability* and *load balance* of the overall system. When a client downloads a data object, it (logically) does so from a single server. In downloading this data object, the client also replicates the associated descriptor from the server. Reliability and balance are improved because the client can now serve queries that match the descriptor; because these are the same queries that the original server could handle, the client can act as an alternate of the original server. Furthermore, depending on the routing topology, this new server can be reachable by peers whose queries could not reach the original server.

However, note that the benefit of replicating a given descriptor decreases with the number of duplicates. If there are N replicas, and the probability of failure is $p << 1$, then the increase in reliability with replica number $N+1$ is $1 - p^N(1-p)$. Clearly, the benefit of replication diminishes very quickly with the number of replicas. A similar argument can be made about the benefits of such replication to load balancing and reachability.

Result #	Terms	Hash Key	Server
1	Mozart Concerto A Major	12fed	123.45.6.7
2	Mozart Violin Concerto	ag231	123.45.6.7
3	Mozart Piano Concerto	3f4a7	34.1.34.1
4	Mozart Clarinet Concerto	12fed	98.12.4.5
5	Mozart Concerto A Major	12fed	85.34.254.5

Figure 1: Ungrouped Results for the Query "Mozart Concerto."

Group #	Hash Key	Terms	Server(s)	Group Size	Precision
1	12fed	Mozart Concerto A Major Mozart Clarinet Concerto Mozart Concerto A Major	123.45.6.7 98.12.4.5 85.34.254.5	3	0.55
2	ag231	Mozart Violin Concerto	123.45.6.7	1	0.67
3	3f4a7	Mozart Piano Concerto	34.1.34.1	1	0.67

Figure 2: Results for the Query "Mozart Concerto" Grouped by Hash Key.

Our P2P file sharing system's metadata management scheme has the dual goals of increasing the *variety* of queries that can be answered as well as the system's *ranking effectiveness*. This is done by heuristically selecting metadata from a group's descriptor, and not from the descriptor of a single server. We assume that the volume of metadata that can fit in a data object's descriptor is limited. The client must therefore be careful in the terms in places in it in order to maximize query effectiveness. We discuss metadata selection heuristics in Section 5.

4.3 Ranking

For each query, many groups may result. To simplify the task of selecting a group, the client ranks each group in terms of its relevance to the query. Our current goal is not to devise new ranking functions, but to consider the feasibility of well-known functions in the current application domain. We discuss various ranking functions in Section 5.

EXAMPLE 1. *Consider a query "Mozart Concerto". The query is sent to all reachable peers, which return results that fulfill the matching criterion. Assume the query returns five results, as shown in Figure 1. (Notice that the results fulfill the matching criterion.) They are grouped according to hash key, as shown in Figure 2. Each group is then ranked by some criterion. In Figure 2, we show the ranking scores using group size and precision (defined in Section 5.3). Notice that each ranking function returns a different high score.*

Assume that the user selects Group 1. By doing so, the user initiates a download for the file with key 12fed. Along with replicating the file, the client must create a descriptor for its local replica. We give two example descriptors in Figure 3. The one on the left is a duplicate of Results 1 (or 5). The one on the right is a random sample of Group 1's descriptor. Notice that this descriptor contains a new combination of terms, and can support new queries.

As mentioned above, the benefit of duplicating a single server's descriptor is greater load balance and reliability. The benefit of the random selection is that the client can serve a previously unsupported request for the data object (i.e., those with some subset of the terms "Mozart Clarinet Concerto A").

5. EXPERIMENTAL RESULTS

We now search for a good metadata distribution technique and ranking function combination. We measure the performance of a combination by the number *successful* queries (i.e., those that lead to the download of the desired data object, described in Section 5.1.1) that the clients perform.

We do not consider traditional IR metrics, such as *precision* and *recall*. Precision measures the percentage of correct results to a query, and is irrelevant because, in our model, any single replica of the desired data object will satisfy the user. For the same reason, recall, the percentage of possible results returned, is also irrelevant in our model.

We first describe our experimental model as well as our P2P system simulator. Subsequently, we describe our ranking functions and metadata distribution techniques. We present experimental results and analyses at the end of this section.

5.1 The Simulator

The design of our simulator is based on observations and analyses of P2P file sharing systems. In the event that relevant design parameters are unavailable, we borrow from work on done on Web information systems and IR.

The major objects in our simulator are terms, data objects, peers, and queries. The **universal set** of terms T that can describe a data object is finite, and each term is assigned a relative access probability based on the accepted Zipf distribution [5]. A random number of terms from T are assigned to each data object's (F_i) universal term **subset** (T_i) based on the initial Zipf distribution. The terms of each data object's universal term subset are then reassigned probabilities according to a Zipf distribution to diversify term usage, as described in [14]. For example, a term that is rarely associated with one data object need not be so for another. We call the set of probabilities that terms will be associated with a data object the data object's **natural (term) distribution**.

In our simulation, we also make the generally unrealistic assumption that terms are independent. For example, the occurrence of "Britney" in a descriptor is independent of the occurrence of "Spears". This is incorrect in general, but is common practice, as it simplifies the model without making it trivial. Note, however, that this term independence assumption is not unique to our work. Such an assumption is heavily relied upon in the probabilistic information retrieval

Duplicating a Server's Descriptor		Randomly Selecting Terms	
Terms	Hash Key	Terms	Hash Key
Mozart Concerto A Major	12fed	Mozart Clarinet Concerto A	12fed

Figure 3: Two Ways of Selecting Metadata from a Group for a Replica.

model in IR.

Each data object is also associated with an access probability, according to a Zipf distribution. This conforms to the access patterns observed for Web objects that were described in [1]. Observations of data object frequency in a P2P system also suggest a high access skew [13].

Initially, a random number of copies of each data object are instantiated, each with a subset of its universal term subset in its descriptor. These copies are assigned to random clients.

There are a fixed number of peers and a fixed number of data objects in the system. At each iteration of the simulation, a random peer is chosen to download a random data object based on the data object' access probability distribution. To do this, the peer generates a query of random length containing a subset of the data object's universal term subset. We assume that query length distributions follow those of Web search engines, and use the empirical distribution presented in [11]. Personal observations of queries in LimeWire's query monitor window seems to corroborate this assumption. Each term in the query is randomly chosen based on the data object's natural term distribution.

The query is routed to a random subset of peers. We do not send the query to all peers because, in practice, only a subset of them is reachable at any time [13]. The peers return results that fulfill the matching criterion to the client (see Section 3).

5.1.1 Client Behavior

If more than one group forms in response to a query, then the client ranks the groups. The client searches through the top-ranked N groups. If the desired result is in that set of N, the will be able to identify and select it. If not, the client will select the highest ranked group. For simplicity, we assume that $N = 1$–the client always selects the single top-ranked group. Although the $N = 1$ assumption is generally a strong one, all else being equal (the client is ambivalent and lets the ranking system make the decision), it is not an unreasonable one. We say that the query is **successful** if the desired data object is downloaded–equivalently, if it is one of the top ranked N results.

Once the data object is downloaded, the user has a probability of manually annotating the data object with some personally chosen terms. These terms are randomly chosen from the data object's universal term subset, based on the natural term distribution. This is the only way that the variety of terms that exists in the system for a data object can increase beyond what exists at initialization. If the user downloads the incorrect data object, then she may misannotate it in this step, corrupting the body of metadata for the data object.

After this is done, the client heuristically copies some of the chosen group's metadata into the replica's descriptor, with the constraint that only a limited number of terms may be copied. The data object is then available for other peers to find (and download) in subsequent iterations of the simulation.

We do not model freeriders or malicious users. Freeriders are users who download, but do not upload data objects. Since they do not contribute any metadata to the system, they do not affect the results. Malicious users are those who may contribute misleading metadata for data objects to the system. These users may affect the rankings, but only marginally. Rankings are based on the aggregate metadata of a group of users, not on the metadata of an individual.

The parameters we use in the experiments are shown in Table 1. The size of the simulation is scaled down to reveal any convergences in the results more quickly. More significant than the scale of the simulation are the relative values of each parameter, such as the total number of possible terms for a data object, versus the number of terms with which each data object is initially annotated. These numbers are based on observations from other studies [12, 13], as well as personal observations. For example, song data objects that appear on Gnutella networks typically have about three or more types of information associated with them from ID3 data: artist, song name, album name, track number, etc. This is reflected in the *Number of terms in initial descriptors* parameter.

We performed forty trials with each set of parameters and report the average results. The 95% confidence intervals generally were well within 4% of the reported mean–the results are statistically significant. However, to simplify the presentation of the main results, we do not present them.

5.2 Goals of the Experiments

The goals of the experiments are three-fold:

1. We first attempt to show that IR can be applied to P2P file sharing systems. In particular, we attempt to show that ranking helps users.

2. Assuming IR is applicable, we attempt to discover a good ranking function.

3. Finally, we then attempt to show the relationship between ranking and metadata distribution; we would like to find a ranking and metadata distribution combination that consistently yields good results.

To achieve these goals, we will test the effects of various combinations of well-known ranking and metadata distribution techniques on our simulator.

5.3 The Effect of Various Metadata Distribution Techniques

In our first set of experiments, our goal is to determine the effect of different metadata distribution techniques on the rate of successful downloads. We consider the following five metadata distribution techniques:

1. Single Server (**server**) - The client creates a descriptor that is a duplicate of the descriptor of a single server.

Parameter	Value or Range
Number of peers	1000
Number of data objects	1000
Number of terms in universal set	10000
Number of terms in the universal term subset of a data object	100-150
Maximum descriptor size for a data object on a peer (terms)	20
Number of terms in initial descriptors	3-10
Number of replicas of each data object at initialization	3
Probability that a peer is reachable	0.5
Probability of client adding metadata	0.05
Number of Terms Added by client	1-5
Query length	1-8, dist from [11]
Number of queries	10000
Number of trials	40

Table 1: Parameters Used in the Simulation.

2. Random (**rand**) - The probability of each *unique* term in the selected group's descriptor of being replicated is uniform. The goal of this technique is to maximize the number of combinations of terms for each data object.

3. Weighted Random (**wrand**) - The probability of each term in the selected group's descriptor of being replicated is proportional to its relative frequency. The goal of this technique is to replicate the term distribution represented by the group.

4. Most Frequent (**mfreq**) - The most frequent terms in the group are replicated. The goal of this technique is to replicate terms that are most likely to appear in a query.

5. Least Frequent (**lfreq**) - The least frequent terms in the group are replicated. The goal of this technique is to replicate terms that are least likely to appear in a general query. The assumption is that these terms help distinguish a data object, and are therefore important.

The impact of a metadata distribution technique is intuitively dependent on the ranking function. This is because ranking function use characteristics of the metadata that are controlled, in this case, by the distribution technique. The ranking functions we consider are:

1. Term frequency (**tf**) - The group whose descriptor contains the most query terms is ranked highest.

2. Precision (**prec**) - The group whose descriptor contains the highest ratio of query terms to total terms is ranked highest.

3. Group size (**gsize**) - The group with the most results is ranked highest.

4. Cosine similarity (**cos**) - The group whose descriptor has the highest cosine similarity to the query is ranked highest.

5. Arrival time (**arrival**) - The group that arrives first at the client is ranked highest. This represents the *non*-ranking of results. **Arrival** represents the low bound for performance.

As shown in Figure 4A, **gsize** is outperforms other ranking function, regardless of metadata distribution technique. Predictably **tf** does poorly due to its tendency to overly weight data objects with large descriptors. **Prec** does poorly because of the small descriptors; a little noise can severely affect a ranking. **Cos** does slightly better, but not significantly better. It relies on unbiased samples of metadata to work effectively, but the matching criterion does not return such samples–query terms are disproportionately represented in query results.

Gsize does best (by at least 20%, as shown in Figure 4A) because it disregards the term distribution of results, as well as the size of the descriptor of an individual result. Instead, **gsize** works by gathering *support* for a particular data object in the form of many matching results. If a particular data object is strongly associated with certain terms, the descriptors of its replicas will also contain them. The matching criterion will therefore return the correct results. Conversely, an incorrect download does little to affect the effectiveness of **gsize** because it only adds at most one misleading replica for an object. We will discuss **gsize** more below.

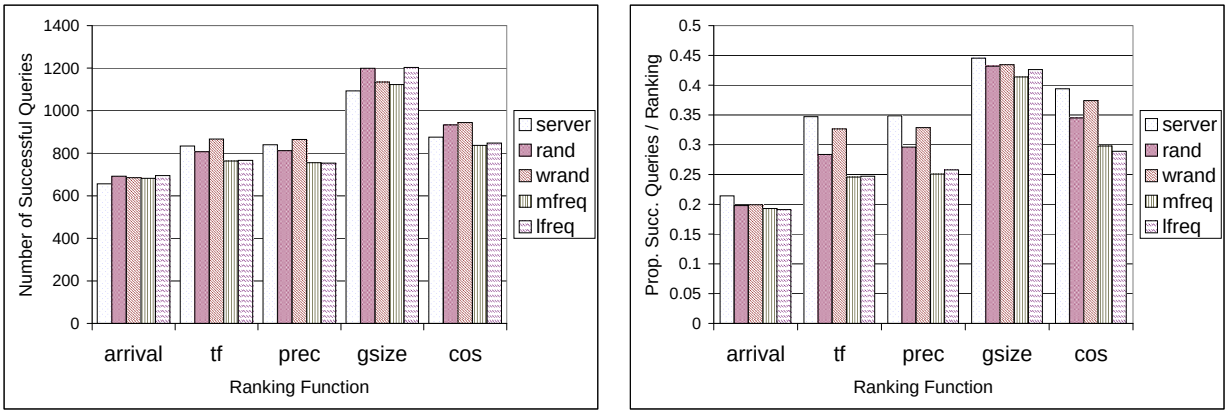
We also consider how active ranking actually is in improving query results. As shown in Figure 4B, a ranking function can contribute from 20% to 45% of the correct results. Ranking is therefore a significant contributor to query result quality.

5.3.1 Relationship between Ranking and Metadata Distribution

The ranking function must be customized for the metadata distribution technique, and consider the matching criterion to yield good query performance. In our experiments, we found that the combination **gsize**/**rand** (short for **gsize** ranking with **rand** metadata distribution) or **gsize**/**lfreq** returned the best results.

The reason **gsize**/**rand** works well is because **gsize** does not consider term distributions in descriptors as mentioned above. It only considers how many results are returned for a particular data object. The likelihood that a data object is returned as a result is maximized if its descriptor contains as great a variety of term as possible, due to the matching criterion. Compared with the other metadata distribution techniques, **rand** is good at maximizing the variety of terms in a descriptor because it considers each unique term equally for replication.

In the *long run*, **lfreq** also maximizes the number of unique terms in a descriptor. This phenomenon is due to the fact



A. Number of Successful Queries.

B. Proportion of Successful Queries Yielded.

Figure 4: Results with Various Ranking Functions / Metadata Distribution Technique Combinations After 10000 Queries.

that the action of **lfreq** constantly changes the set of terms that qualify as infrequent. Furthermore, **lfreq** copies each term into a descriptor only once. **Lfreq** works best when there is little metadata (which is true in this case). Otherwise, the descriptors that **lfreq** creates will match very few queries.

Uniformly selecting metadata may create a lot of *noise* in the result set. That is, if all results contain many terms, it may become impossible to distinguish them. In practice, however, a term that is strongly associated with one data object may be only weakly associated with another. When such terms appear in queries, they act as effective filters of unwanted results, boosting the rank of wanted results:

EXAMPLE 2. *The term "pop" is very likely to be associated with music files in general. However, its likelihood of being associated with a particular artist is unique. Consider the musicians Michael Jackson and Michael Bolton; in the former case, the term "pop" is much more strongly associated. Therefore, the query "Michael pop" probably refers to the former. The matching criterion will likely return more Michael Jackson songs than Michael Bolton songs, as expected.*

As shown in Figure 4A, when **cos** is the ranking function, **wrand** is the best metadata distribution technique. This is expected because, as shown in Table 2, **wrand** does a good job of maintaining a body of metadata whose distribution matches that of the natural term distribution. This is vital to the performance of **cos**. Note that although **server** does a good job at maintaining the natural term distribution (according to its score in Table 2) unlike **wrand**, it does not increase the variety of terms in the descriptors.

Cos/**rand** does slightly better than does **cos**/**server**, **cos**/**lfreq** and **cos**/**mfreq** because **rand** gives a slight preference to more frequently occurring terms during metadata distribution. This effect is also suggested in Table 2. This happens because query terms, which are strongly associated with a data object, are more likely to be replicated using **rand** due to its interaction with the matching constraint.

5.4 Searching the Top N Results

In our basic experiments, we assumed that the user automatically selects the highest ranked group. This models a user who trusts the ranking function, and does not bother inspecting the metadata. We also consider the case where the user performs a search of the top N results. In this case, we assume that the user can identify the desired correct data object among the top N if it exists. If the correct data object is not in the top N, the user just downloads the highest ranked data object.

server	rand	wrand	mfreq	lfreq
0.69	0.66	0.69	0.64	0.63

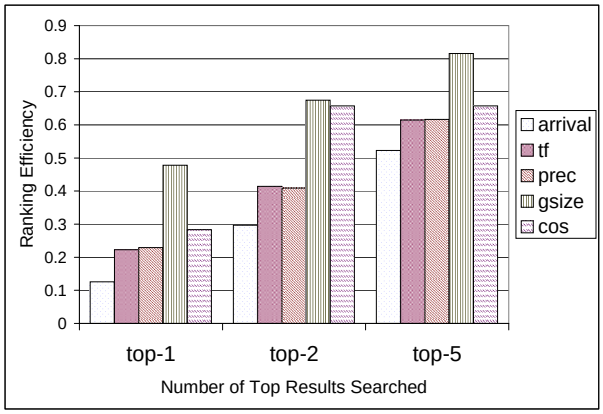
Table 2: The Cosine Similarity of the Final Metadata Distribution in the System and the Natural Term Distribution after 10000 Queries. Results are averages over all ranking functions.

We define **ranking efficiency** as the ratio of the number of times a group is correctly ranked and the number of times ranking must be done. As shown in Figure 5A, the ranking efficiencies of all ranking functions increases with N *and* are always greater than the ranking efficiency of **arrival**. This suggests that the ranking functions are generally effective in ordering results based on their relevance to the query. The knee in the **cos** graph indicates that it is reasonable at ranking groups, but fails to consistently identify the most relevant one.

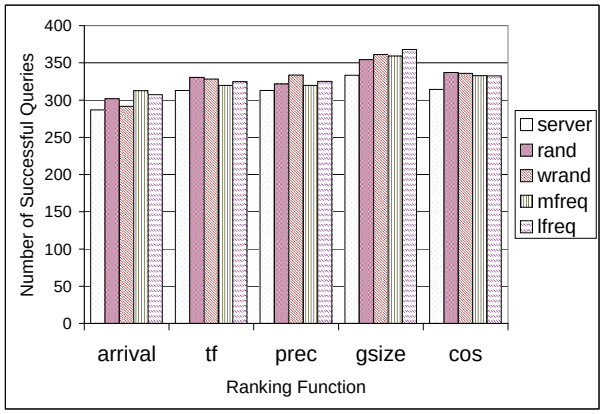
5.5 Query Length

We now consider the effect of query length on the quality of ranking. Increasing query length gives the ranking function more information, so we expect better ranking performance. However, this comes at the tradeoff of fewer results. For simplicity, assume that each term has a probability $p \leq 1$ of appearing in a data object, and that terms occur independently and are uniformly distributed. Due to the matching criterion, the proportion of results that are returned for a query of length L_Q is p^{L_Q}.

We doubled and tripled our basic query lengths. When we doubled the query length, we increased the ranking efficiency for **gsize**/**rand** (for example) from 0.49 to 0.53, but decreased the average number of results per query from 4.4 to 0.3. The net result is a decrease in the potential number of successful queries by 75% as shown in Figure 5B. Tripling



A. Ranking Efficiency when Inspecting the Top N Results. Averages of each ranking function reported.



B. Number of Successful Queries with Doubled Query Length.

Figure 5: Top N and Increased Query Length Results after 10000 Queries.

the query length showed a similar phenomenon, but to a greater magnitude.

5.6 Summary of Results

We now revisit the questions concerning IR in a P2P file sharing environment. The most fundamental question is whether or not IR can work in such an environment. To argue this point, we cite the ranking activity suggested in the experiments, particularly in Figures 4A and 5A. From the former it is clear that ranking contributes to quickly finding desired data. From the latter, is it clear that the rankings beyond the first object are consistent.

We now consider the design of a P2P IR system. The first question is which ranking function to use. From our experiments, **gsize** performed best, regardless of the metadata distribution function, with a range of 1060 to 1213 successful queries out of 10000 attempts. **Cos** ranking was second best, but significantly worse than **gsize**, with 843-963 successful queries. As a corollary, in the special case where no metadata distribution takes place (i.e., **server**, which is used in practice), the best ranking function to use is **gsize**. See Figure 4A.

The second question is which metadata distribution technique to use. The best techniques maximize the variety of terms in descriptors. This is done in distributed environments by randomizing the metadata distribution. As shown in Figure 4, **rand**, **wrand**, and **lfreq**, which randomize descriptors, generally outperform both **server** and **mfreq**, which do not. The particular metadata distribution technique, however, depends on the ranking function used. With **cos** ranking, **wrand** metadata distribution is best, and with **gsize** ranking, **rand** metadata distribution is the best. At the same time, **mfreq** and **server** generally perform poorly, regardless of the ranking function because they create few or no new combinations of metadata terms in descriptors.

6. CONCLUSION

Our goal was to reveal how the unique characteristics of practical P2P file sharing systems (i.e., short queries, little metadata, high churn rate) impact their ability to perform information retrieval. The factors we considered most significant in query performance are the query matching criterion, metadata distribution technique, and ranking function. Based on our experiments, descriptors should contain as many unique terms as possible. This is realized in our experiments by the **rand** and **lfreq** metadata distribution techniques. Given these metadata distribution techniques, **gsize** ranking works best due to its lack of consideration for term distributions in descriptors.

Cosine similarity (**cos**), a well known IR ranking technique, can best be implemented using **wrand** metadata distribution. However, due to the characteristics of P2P systems mentioned above, **cos**/**wrand** perform significantly worse than does **gsize**/**rand**. Other IR techniques (e.g., TF-IDF, not discussed in this paper) also suffer from similar problems in this environment.

We are currently devising novel ranking and metadata distribution techniques that are customized for the P2P enivronment. These algorithms will have to consider the instability and paucity of metadata in the system.

We are also currently building a prototype of our P2P file sharing system based on freely available Gnutella source code. Once our prototype is stable, it will be published on the author's Web site. Releases will be in stages. The first will contain simple ranking and metadata distribution functionality, but subsequent versions will contain more sophisticated IR techniques, as well as some statistics gathering functions.

7. REFERENCES

[1] M. Crovella and A. Bestavros. Self-similarity in world wide web traffic: evidence and possible causes. *IEEE/ACM Trans. Networking*, 5(6):835–846, 1997.

[2] Free Peers, Inc. Bearshare technical faq. Web document, 2004. www.bearshare.com/help/faqtechnical.htm.

[3] P. G. Ipeirotis and L. Gravano. Distributed search over the hidden web: Hierarchical database sampling and selection. In *Proc. VLDB*, pages 394–405, 2002.

[4] jackie@audiograbber.com us.net. Audiograbber home page. Web Document.

[5] D. E. Knuth. *The Art Of Computer Programming*, volume 3:Sorting and Searching. Addison-Wesley

Publishing Company, second edition, 1975.

[6] LimeWire, LLC. Gnutella protocol 0.4. Web Document, 2004. www9.limewire.com/developer/gnutella_protocol_0.4.pdf.

[7] B. T. Loo, J. M. Hellerstein, R. Huebsch, S. Shenker, and I. Stoica. Enhancing p2p file-sharing with an internet-scale query processor. In *Proc. VLDB*, Toronto, 2004.

[8] J. Lu and J. Callan. Content-based retrieval in hybrid peer-to-peer networks. In *Proc. ACM Conf. on Information and Knowledge Mgt. (CIKM)*, pages 199–206, Nov. 2003.

[9] W. Meng, C. Yu, and K.-L. Liu. Building efficient and effective metasearch engines. *ACM Comp. Surveys*, 34(1):48–84, Mar. 2002.

[10] N. I. of Standards and Technology. Sha1 version 1.0. Web Document, 1995. www.itl.nist.gov/fipspubs/fip180-1.htm.

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

[12] M. Ripeanu and I. Foster. Mapping the gnutella network: Properties of large-scale peer-to-peer systems and implications for system design. In *Intl. Wkshp. on P2P Sys. (IPTPS)*, number 2429 in LNCS, Mar. 2002.

[13] 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.

[14] 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.

[15] A. Singla and C. Rohrs. Ultrapeers: Another step towards gnutella scalability. Technical report, Limewire, LLC, 2002. rfc-gnutella.sourceforge.net/src/Ultrapeers_1.0.html.

[16] K. Sripanidkulchai, B. Maggs, and H. Zhang. Efficient content location using interest-based locality in peer-to-peer systems. In *Proc. IEEE INFOCOM*, 2003.

[17] C. Tang, Z. Xu, and S. Dwarkadas. Peer-to-peer information retrieval using self-organizing semantic overlay networks. In *Proc. ACM SIGCOMM*, Aug. 2003.