The Design of PIRS, a Peer-to-peer Information Retrieval System

Wai Gen Yee and Ophir Frieder

Database and Information Retrieval Laboratory
Department of Computer Science
Illinois Institute of Technology
Chicago, IL 60616
yee@iit.edu, ophir@ir.iit.edu

Abstract. In this paper, we describe the design of PIRS, a peer-to-peer information retrieval system. PIRS satisfies many of the goals of P2P computing in a way that other P2P IR systems do not: It does not require the centralization of data onto specially designated peers, and is therefore applicable to a larger application environment. We explain our design decisions, analyzing their potential benefits and shortcomings. We then demonstrate PIRS's search performance via simulation.

1 Introduction

Many of today's peer-to-peer (P2P) file sharing systems were initially conceived as successors to Napster, which was used primarily for the exchange of music. As such, they are designed to allow simple annotation of files, including the artist and song title.

As long as a file's metadata are well-known, searches are simple. A query matches a file if its terms are substrings of the metadata's terms. For example, a query "Pumpkins" will return a file annotated by "Smashing Pumpkins 1979." However, even though the song, 1979, is one of Smashing Pumpkins greatest hits, the alternative query "Smashing Pumpkins Greatest Hits" will not return that file.

The problem described above illustrates a limitation of P2P search: It requires the user to know the exact metadata associated with the file in order to perform a successful search. This is problematic for the naive user, who is unaware of annotation conventions, or for a user not looking for a particular song, but for a particular type of music (e.g., "songs from bands from Chicago"). This problem is exacerbated by the fact that many song files are automatically annotated using Web databases, such as freedb.org. Such annotation results in the identical annotation of all copies of a particular song, and gives users disincentives to make their own annotations.

The goal of this paper is to describe how information retrieval (IR) can help alleviate this problem in a P2P environment. In our analysis, we spend some time describing the characteristics of P2P systems, and the degree to which existing

P2P systems possess them in Section 2. In Section 3, we informally describe the P2P file sharing model. We describe the design of PIRS, our **P**2P **IR** **S**ystem, in Section 4, and then discuss its pros and cons compared with other P2P IR systems in Section 5. We describe our PIRS system simulator and present some experimental results in Section 6. We make concluding remarks about the work presented in this paper, and discuss the future of P2P IR in Section 7.

2 Limits of Current Work in P2P Information Retrieval

2.1 Characteristics of P2P Systems

As described in [1], P2P systems are characterized by low maintenance overhead, improved scalability and reliability, synergistic performance, increased autonomy, privacy, and dynamism. P2P systems are inexpensive to maintain and have good scalability because they use the full resources of all participants, who function as both clients and servers. They are more reliable, because the failure of any one peer does not disable the entire system. They offer synergistic performance because peers can utilize resources (e.g., data, CPU cycles, disk space) that are underutilized by others. They are dynamic, allowing peers to autonomously join and leave the system, and thereby changing the types of resources offered. They offer privacy because P2P networks lack central authentication servers. P2P systems are therefore ideal in environments populated by many autonomous users with dynamic needs and surplus resources.

These characteristics distinguish P2P systems from previous technologies, such as distributed computing, and ad-hoc communication. Distributed computing refers to computing on a platform where resources are located on many physically distinct platforms. Unlike P2P systems, these resources may be highly integrated and interdependent. Ad-hoc communication refers to a communication platform in which a client can automatically join an existing network. Ad-hoc networking deals with the lower-level communication infrastructure on which P2P systems can be built. P2P computing is therefore a unique paradigm that creates new possibilities for information technologies.

2.2 The Limits of Information Retrieval Using Web Search Engines

The Internet offers a medium by which everyone can easily gather, and share information. Today, the dominant paradigm for sharing information is via the Web. Organizations set up Web servers that host Web sites where they can publish information. Individuals also have a chance to publish information through personal Web pages or *blog* pages. To access this information, users simply type in the appropriate URL into a Web browser.

There is a gap, however, in bringing together information publishers and consumers–how exactly does one find the appropriate URL that points to desired information? Today, this gap is bridged by Web search engines, such as Google. A consumer enters some relevant terms into Google, which returns heuristically-defined best matches.

The problem with relying on Google to find published data is that publishers must wait for Google to *crawl* their Web pages before they appear in any search results. Because crawling occurs only periodically, material published today will not be found for some time. Another problem is that Google caches indexed content. Once this happens, publishers lose control over the dissemination of their material [2]. Furthermore, the results returned by search engines can be suspect. For example, rankings can be influenced by commercial interests [3]. Finally, centralizing query services limits the scalability and reliability of the search engine: A single server can only index so much content (Google claims to index 4 billion Web pages, which is considered a small fraction of those available), and also is a single point of failure. A more relevant example is Napster, whose centralized architecture made it easy prey for legal attacks.

Recent P2P file sharing systems that focus on file transfer, such as BitTorrent [4] suffer from the same problems as Web search engines. BitTorrent is different from Gnutella in that the former focuses on download performance, whereas the latter focuses on search. BitTorrent allows a client to download a single file from multiple sources simultaneously, thereby improving performance, and distributing load. However, it relies on Web search engines to provide search capabilities for shared content, and therefore has all the problems discussed above.

P2P systems solve many of these problems. Because queries are sent directly to data sources, results are up-to-date and reflect the currently available data. Upon receipt of results, the peer can use custom algorithms to rank them. This eases their perusal as users have more trust in their rankings. Finally, there is no single point of failure with P2P systems. A query likely returns results even with the failures of multiple nodes [5].

2.3 Current Peer-to-peer Information Retrieval Systems and their Limits

Work on P2P information systems has focused on either bandwidth efficiency, or the development of unifying data models. The PeerSearch system [6] is built on top of the CAN routing infrastructure [7]. CAN places content in a P2P network based on its hash value. PeerSearch proposes to create a distributed index, which is partitioned and similarly placed on a network. This deterministic placement of content improves bandwidth efficiency by constraining the way a query is routed. (The original version of Gnutella, in contrast, floods queries over the network [8].) In [9], the authors take a similar approach. They assume a hybrid networking architecture where some the peers that have been deems more reliable and capable act as servers. These servers, besides routing queries, also store metadata statistics, such as term frequencies, that are used by traditional IR algorithms.

Other systems, such as Edutella [10] and PeerDB [11] propose data models that standardize the way data and services are published and queried.

While these systems have much potential, they are limited due to the constraints that they put on the infrastructure and applications. The PeerSearch

system works best in an environment where peers are reliably connected to the Internet. This is necessary because shared content is centralized in certain peers based on their hash values. The loss of a peer results in the loss of all its associated content. Furthermore, it takes control of data placement out of the users' hands. These characteristics violate the principles of P2P systems that are described in Section 2.1. A solution to this problem is to replicate content by applying multiple hash functions on content. This is problematic as well, because it increases both the amount of data every peer must maintain and network traffic as well. Notably, no work we know of has been done on P2P information retrieval in highly environments where peers frequently join and leave the network.

Edutella and PeerDB focus more on standards than on information retrieval. Standardization, however, tends to raise the bar for entry into a network because it forces users to do more work to publish content. This has the effect of limiting the amount of data that are published, thereby reducing the network's overall usefulness [12].

Note that it is not our goal to be purist in P2P system design. The success of Napster (in terms of market impact and user satisfaction) demonstrates that, under certain conditions, there is no need. At the same time, pure P2P systems were shown to have scalability problems [13], which can be alleviated by the use of a hybrid architecture [14]. However, the fact that a system works without being purely P2P does not mean that it might not work better if it were so.

3 Model

In a typical file sharing model, each peer (which we may refer to as a client or a server, depending on the context) maintains a set of shared files. Each file is annotated with metadata *terms*. The particular terms associated with each *instance* of a file (the file's *metadata set*) is user-tunable.

Users create *queries* in order to find files in the P2P system. A query consists of terms that a user thinks best describe a file. These queries are routed to reachable peers. (Queries generally do not reach all peers due to network conditions or details of the routing protocol.) Returned are pointers to instances of files that match the query, and the file's metadata set. The matching criterion is for all the query terms to be substrings of some term of the file's metadata set.

Users study the returned metadata sets to decide on the file to download. Once the user makes her selection, she downloads the file by selecting its associated file pointer. The client follows the pointer, downloads the file, and then becomes a server for an instance of that file.

Note that although the discussion in this paper uses music file sharing as an application, it also applies to other applications. For example, an HTML document is also a file that can analogously be annotated with metadata in the form of `META` tags. The terms in the `META` tags can be tuned independent of the "content" of the HTML document.

4 The Design of PIRS

4.1 Goals

Our goal is to design a P2P IR system that focuses on client behavior and is fully distributed. The system must make little or no assumptions about the underlying communication infrastructure and the behavior of servers (i.e., other peers). For example, CAN routing and PeerSearch (mentioned above in Section 2) tacitly assumes that the network is stable and servers are reliable. Consequently, although these systems have potentially excellent performance, violating either of these assumptions results in the loss of either queries or data. In this light, these systems tend to fall somewhat between the categories of distributed and P2P systems.

A system that does not make assumptions about the communication infrastructure and behavior of peers avoids these problems. The obvious questions to ask therefore are:

1. How well would such a P2P IR system work? For example, IR requires global statistics about the available data for effective ranking. In a highly dynamic environment, such statistics are hard to yield. Furthermore, even if such data were available, would it be possible to implement IR ranking algorithms in a P2P application? The question here is about performance in terms of query result quality as well as computational complexity.
2. Could such a system adapt to changes in system conditions? Making no assumptions in designing a P2P system may be too conservative an approach. In some cases, the network and peers are capable and reliable. Can the P2P system take advantage of this condition, if available? Gnutella's Ultrapeer architecture demonstrates adaptability; it conserves bandwidth given a stable environment, but also works (albeit less efficiently) in an unstable one [14].

Our goal is to answer these two questions. To do this, we describe the design of the PIRS P2P IR system. In doing this, we will highlight the complexities of applying IR techniques to a P2P environment.

4.2 Overview

PIRS is designed to combine the search capabilities of information retrieval systems with the dynamic network management of P2P systems. It works by managing metadata in such a way as to *gradually increase the variety* of queries that can be answered for a given file. This is done by adapting the annotation of a particular file to match query patterns. PIRS accomplishes its goals in three ways:

1. Metadata collection (Section 4.3) - Collect as much metadata as possible for a file, using various means. Increasing the amount of metadata increases the likelihood that a query will find matches.

2. Metadata distribution (Section 4.4) - Heuristically replicate metadata from other peers for a given file. By sharing metadata from multiple peers, the variety of queries that can be matched for a given file increases.
3. Metadata use (Section 4.5) - Utilize IR techniques to rank results, disambiguating them, and thereby improving the likelihood of a correct download.

The processes of metadata collection, distribution, and use work together to improve the search capabilities of PIRS. Ostensibly, they can work independently to improve search, but with diminished benefits. For example, IR ranking techniques alone can be incorporated into Gnutella, without PIRS's metadata distribution techniques.

By design, PIRS is simple to incorporate into many existing P2P protocols. This is a consequence of its functionality being concentrated on client behavior, and its independence from networking infrastructure. Many existing P2P protocols focus on aspects of query routing, which is independent of PIRS's functionality. Consequently, PIRS can be built on top of many of today's popular P2P file sharing applications, such as Gnutella and FastTrack. We will discuss this in more detail in Section 5.

PIRS Versus Other P2P IR Systems The major difference between PIRS and other P2P IR systems is that PIRS treats metadata as a dynamic resource that should be managed collectively by all peers. Effective management of metadata improves query result quality. The inspiration of this work stems from the notion that, from a client's perspective, the P2P network is a repository of files, each of which is described collectively by a *body of metadata*. The better a file's body of metadata describes the file, the easier the file should be to find.

Current P2P IR systems do not have this perspective. They treat each download as an individual transaction, without regard to how it (the download) affects the file's body of metadata. The download of a file generally also results in the replication of that file's metadata from a particular server. The downloading client becomes a server for the file, but with marginal benefit, because the clients it serves are exactly those which the original server serves. The new server's role in the network is largely redundant.

4.3 Metadata Collection

Metadata collection the process by which a file is annotated with identifying terms. In this section, we describe how metadata collection is typically done in commercial `P2P` systems. We also describe a unit of metadata that PIRS exploits for good performance.

Metadata terms are directly used for query matching. It is therefore important to build into PIRS effective means of annotating files. One of these means include creating an easy to use user interface, which encourages users to add metadata. Other means include automatic, user automatic annotation and metadata foraging.

Recent versions of P2P file sharing systems offer templates that help a user annotate certain types of files, such as audio files, using special application-specific fields [15]. These templates structure metadata, potentially increasing the query matching possibilities.

Much metadata are also automatically foraged from Internet sources. For example, when *wav* files are *ripped* from commercial compact disks, the ripping software automatically collects ID3 metadata (e.g., title, artist) [16] for it from Web sites such as freedb.org. Other metadata are *intrinsic* to the file. Such metadata include the size of the file, its filename, and the last time it was accessed. Making these metadata available for querying requires some simple programming.

Finally, some systems automatically derive useful metadata from the actual file. BitTorrent, for example, generates a unique hash key for each file, which can simplify its search and be a means of validating the file's contents [4]. A hash key can also be used to group files that are returned by queries.

PIRS uses a file's hash key for validating and grouping files. Such use of the hash key has not been universally adopted. LimeWire's Gnutella groups by filename, file type, and file size [17]. BearShare and eDonkey only use hash keys to authenticate files.

One problem with requiring all files to be annotated with a hash key is its computational cost. This problem has been acknowledged by BearShare, which claims that computing keys in a background process takes only 25% of a CPU's cycles [18]. Hash keys can also be computed while a file is being downloaded, extracted (if it is compressed), or ripped. Piggybacking these processes amortizes the cost of computing the hash key.

Maintaining a hash key for files also does not hurt PIRS's compatibility with existing P2P file sharing systems. It would be treated as another generic unit of metadata by a peer that did not realize its significance.

4.4 Metadata Distribution

Collected metadata for a file on each node on which it is replicated constitutes a collective body of metadata for that file. The larger and more descriptive this body of metadata, the more likely a query will result in relevant results. Metadata collection alone, however, must be supplemented with good metadata distribution techniques.

There are two reasons for this. First of all, if metadata are not distributed, then the system becomes vulnerable. If all metadata are concentrated on a single peer (e.g., as with Napster), the system becomes unusable if this peer is unreachable. This vulnerability violates a basic principle of P2P systems.

Second, data that are not distributed properly could leave correlations in term occurrences, which limit the degree of query matching. For example, assume there are two metadata ripping systems for song files: one extracts the album name and the song's track number, and the other extracts the album's label and year. If files were only annotated using one of these two rippers, then a query

containing an album (corresponding to the first ripper) and a year (corresponding to the second ripper) cannot be properly matched.

PIRS distributes metadata in a way that avoids the problem just described. PIRS attempts to do three things: maximize the amount of metadata associated with each file; remove correlations in query terms; make frequencies of a term's occurrence in a body of metadata reflect its frequency of occurrence in queries for it.

In designing PIRS, we assume that there is a potentially large, but finite set of terms associated with each file. Of these terms, some are more strongly associated with the file than are others. The stronger the term's association, the more likely it will appear in a query. For example, a song's title is very strongly associated with a song file, but the song's publisher is less so. The aggregate metadata distribution in a P2P system should reflect this tendency in order to improve query matching.

Metadata distribution occurs during queries that are followed by downloads. The first step in this process is to group the query results (including their metadata sets and pointers) by hash key. The user browses the groups of metadata to select the file for download. Because the selected group may contain a large set of metadata, the user may only select a subset of the metadata. This refer to the selection as metadata distribution. We consider five ways of distributing metadata:

- Server terms (**trad**) - The client selects the metadata that exist on the single server from which it downloads. This is the solution that is commonly used in today's P2P file sharing systems. It is notable for its simplicity.
- Most frequent terms in the group (**mfreq**) - The client selects the terms that occur most frequently in the group. The justification for this approach stems from the assumption that, because these terms appear so much, they are strongly associated with the file, and therefore most likely to occur in queries.
- Least frequent terms (**lfreq**) - The client selects the terms that occur least frequently in the group. The usefulness of this approach is that these terms help distinguish this file from others. It also balances out the term distribution.
- Random terms (**rand**) - The client randomly selects terms from the group, maximizing the number of term combinations.
- Random terms based on freq (**randfreq**) - The client randomly selects terms from the group weighting more frequently occurring terms proportionately higher. Like **rand**, this technique also increases the number of term combinations, but gives preference to more commonly occurring terms.

In the last four techniques, **mfreq**, **lfreq**, **rand**, and **randfreq**, the client selects as many metadata terms as it can for each file.

4.5 Metadata Use

By judiciously distributing metadata, the client increases the variety of metadata combinations that exist in the network. This has the effect of increasing the

variety of queries that can be satisfied and the number of results returned per query. This increases the amount of work the user must do to find the correct file. Ranking lightens this burden by speeding up the time required to find the correct file.

Good ranking is also important in keeping the body of metadata for each file correct. A poor ranking function may induce a user to download the incorrect file, and then misannotate it with terms that are meant for the intended file. This incorrect metadata may mislead other peers, leading to further incorrect metadata.

We consider five ranking techniques. Some of these techniques are classical IR techniques, and some are unique to P2P file sharing:

- Group size (**GS**) - The number of results in a group. A large GS indicates that either a particular file has large support for satisfying a query, or that the file is generally popular, and is therefore something desirable anyway.
- Term frequency (**TF**) - Counting the number of times query terms appear in a file's metadata. Terms that occur frequently in metadata sets likely represent the contents of the file.
- Precision (**prec**) - Dividing TF by the total number of terms in the group. Precision adjusts for problems with TF caused by large metadata sets.
- Inverse term frequency (**ITF**) - Starting with TF, but weighting each term by the inverse of its frequency in the set of results. Less frequent terms get more weight, because they are assumed to be better discriminators of a file's content.

These algorithms may be used by PIRS with some modification. For example, GS, TF and precision are straightforward ranking techniques. ITF (based on *idf* ranking [19]) should use global instead of local knowledge about term frequencies, but global knowledge is unavailable to any individual peer. ITF may also be computationally expensive. Other ranking techniques, such as PageRank [20] are inapplicable to the P2P environment due to the lack of hyperlinks.

4.6 Implementation Issues

One of the characteristics of P2P file sharing systems is that results of user queries arrive asynchronously. This is a consequence of the distributed, independent, and heterogeneous nature of the data sources. A peer can vary in its location, computational capability, network connectivity, and so on. Results therefore arrive over a period of time. In LimeWire's implementation of Gnutella, this period is about three minutes, after which further results are ignored.

Results are displayed to the user as they arrive. This is an interface design decision that gives the user feedback on the progress of the query. It also allows the user to pick a file to download as soon as she sees it displayed. These characteristic complicates PIRS's use of IR techniques.

To be helpful to the user, results must be organized in a reasonable manner– not simply in the order of arrival. Specifically, similar results must first be

grouped, reducing the number of results the user must examine. The groups must then be ranked in order to allow the user to more easily find the relevant query.

In traditional IR, rankings are computed on static data sets, and subsets of the resultant rankings are displayed to the user on request. This is intuitive behavior, as one would expect all necessary information to be available to an algorithm for it to make a decision as to the relative value of a file.

The ranking problem is slightly more complicated in the P2P case because of the asynchronous arrival of results, the need to display results to the user in real-time, and the lack of global statistics on data. Instead of computing a monolithic set of rankings, the ranking algorithm must be able to update the current rankings as new results arrive in a reasonable time. The resultant rankings may also be spurious due to the fact that no global statistics are available (as we will discuss below).

LimeWire's implementation of Gnutella is very popular. We therefore use it ranking and grouping technique as a model, comparing to it the performance of PIRS. LimeWire performs grouping using filename, file size, and file type. The worst-case complexity of LimeWire's grouping algorithm is $O(N^2 \log(N) KM)$, where N is the number of results, K is a similarity metric (edit distance of two filenames), and M is the length of the filename. Files are first sorted by size. Files with similar sizes are checked for similarity by their filenames and file types. LimeWire does not perform ranking [17].

PIRS's interface displays groups of files, dynamically updating the memberships of these groups. As results arrive, the rank of each group changes. The ordering of the groups in the user interface, however, does not. This is a design decision that allows the interface to maintain some stability. We consider the automatic reordering of groups caused by updated rankings disruptive to the user's experience. On the other hand, the user has the option of explicitly requesting that the groups be reordered once based on current rankings. We will now demonstrate how PIRS grouping and ranking can be implemented in a complexity that is comparable to that of LimeWire grouping.

As mentioned in Section 4.5, term frequency (TF) is one of the most basic algorithms for ranking results. Its measure is the number of times a query term occurs in metadata. The problem with this metric is that it disproportionately weights very frequent terms. For example, the frequent term (e.g., *classical* for describing music) is less indicative of content than a rare one (e.g., *baroque* for describing music). Inverse term frequency (ITF) attempts to solve this problem by dividing each term's contribution to the rank of a group by that term's overall frequency over all groups [19].

There are two challenges to implementing ITF in PIRS. The first challenge is to compensate for the lack of necessary statistics in the P2P environment. The second is to implement it so that it efficiently handles asynchronously arriving results, as described above.

ITF requires information on the frequencies of all terms that exist in all metadata sets in the network in order properly function. The problem is that, in

a dynamic P2P environment, it is impossible to gather this information because the set of available metadata is always in flux as peers join and leave the system. Past works have claimed that precise frequency statistics are not necessary for reasonable accuracy [9]. With this assumption, we work with the information that is available–the metadata that accompanies the results a query–and make estimates about term frequencies based on this asynchronously growing metadata sample.

Incorporating ITF ranking in Limewire's Gnutella is computationally complex. Each time a result arrives, each of its metadata's f_D terms must be counted, and the rankings of N groups must be updated. Updating a ranking for a query term q requires updating the contribution of q to a group. This requires $O(1)$ work. The complexity of updating the ranking with the arrival of each result is therefore $O(f_D N)$ times the cost of making the groups, for a total complexity of $O(N^3 \log(N) K M f_D)$.

PIRS simplifies the process of grouping and ranking. As a result arrives, it is grouped based on its hash key. This requires $O(\log(N))$ time for each result, or $O(N \log(N))$ for all N of them. Each term in the result's metadata is hashed into a table that keeps track of its frequency in the group as well as its total frequency. Because each term in the metadata may affect rankings, the total complexity of updating a ranking due to the arrival of a result is $O(N f_D)$, and the total complexity to group and rank is $O(N^2 \log(N) f_D)$.

We can reduce the complexity of grouping and ranking by only occasionally updating the rankings in response to result arrival. For each query term, q_i, we wait until its frequency, f_i, is 2^j for some integer $j \geq 0$ before updating its contribution to a group's rank. For example, if a query contains the term "pop," then we only update the ranks of groups for the 1st, 2nd, 4th, 8th, 16th, etc., occurrences of "pop" over all the results. If "pop" occurs a total of f_i times, then the rankings will be updated $\lceil \lg(f_i) \rceil + 1$ times instead of f_i times as a result of this term. The justification for this method is that, as f_i increases, the marginal effect of q_i on the rankings decreases. Increasing f_i from 1 to 2 halves q_i's impact on rankings, but increasing f_i from 2 to 3 only decreases its effect by a factor of two-thirds.

The complexity of grouping and ranking using ITF is now significantly improved. If the number of terms in the query is q_T, then the average frequency of a query term is at most $\overline{f} = \frac{N f_D}{q_T}$. The complexity of ranking is therefore $O(N \log(N) + N f_D + N q_T(\lceil \lg(\overline{f}) \rceil + 1)$. The first term in the expression represents the complexity of the grouping of results based on hash key (a sort operation). The second term represents the complexity of updating the frequencies of each term. The third term represents the complexity of updating the rankings for each of the N groups $\lceil \lg(\overline{f}) \rceil + 1$ times for for each of the q_T terms.

The complexity expression contains separate terms for grouping and ranking because these operations are independent. This complexity is clearly a significant improvement over the unoptimized ranking algorithm that uses LimeWire grouping. But it is also comparable to LimeWire's grouping algorithm without ranking.

5 Pros and Cons of PIRS's Design

One of the features of PIRS is that its functionality is concentrated in the application layer. Consequently, regardless of the reliability of the network and the reachability of any one client, PIRS can function. This means that PIRS can work with any routing protocol devised for P2P networks such as Gnutella routing. The problem with Gnutella routing is that each query floods the network.

A solution to flooding is to incorporate hash-based routing, such as CAN, into PIRS. Each file's metadata set (as well as user queries) is treated as a vector, where each element represents a term's frequency. The vector is hashed into a key that determines its placement (routing) in the network using CAN techniques. This is exactly what is done in PeerSearch [6]. We do not standardize this feature in PIRS for reasons stated in Section 2.

Another feature of PIRS is that it is backward compatible with Gnutella, and that Gnutella is forward compatible with it. This is true PIRS-specific activity is isolated in its *client* behavior. Server behavior is equivalent in Gnutella and PIRS. It can also be argued that PIRS query results are no worse that standard Gnutella results, and that a PIRS peer does not adversely affect the performance of Gnutella peers.

PIRS focus on application level functionality can also be a limitation. The fact that its functionality does not make assumptions about the network means that it may miss out on certain performance optimizations. Specifically, PIRS performs best compared to other systems when the network is unreliable. The question is whether PIRS can be made to work better in more stable conditions.

Analogous to how modern Gnutella networks designate some peers as Ultrapeers to make routing more efficient, PIRS can designate some peers as statistics servers. Statistics may be collected actively or passively, and may include routing information, query information, and information about the repositories of nearby peers. The system described in [9] does just this for P2P information retrieval, and we are also incorporating such functionality into future versions of PIRS.

6 Modeling a PIRS Network

We designed a simulator that captures the behavior of a PIRS system. It simulates queries and downloads from a query to the general reachable peer population. Because PIRS makes no assumption about network capability, aside from the fact that only a subset of peers is available at any time, it is not necessary for our simulator to simulate a network topology. We also do not consider the response time of the network. The user is assumed to make a decision when all responses have arrived at the client.

We assume that a user always downloads the file that is ranked highest. By doing this, we assume a significant role for the ranking algorithm. Related to ranking is the ability of the user to discriminate highly ranked, yet irrelevant results. We model this by assuming that the user can peruse all metadata associated with a file, much in the way the modern Web search engines give a hint

at the Web site that is pointed to by using some quoted text. If the user sees metadata that contradicts her expectation of the file (e.g., the term *Madonna* for a Michael Jackson song), then she will automatically reject the file. This is how we simulate a user's behavior in overriding the suggestion of the computer by using human judgment.

File-term mapping is done as described in [21]. Each file is annotated with a set of terms from a dictionary of words associated with it, such that the overall distribution of term frequencies of all files is Zipf. The likelihood that each file is downloaded is also Zipf. These distributions conform to the notion that some words naturally occur more frequently than others, and that some songs are more popular than others. Finally, queries are constructed using terms from the file's dictionary.

6.1 Some Experimental Results

In these experiments, we demonstrate the effect that different metadata distribution techniques have on the rate of correct downloads. We fix group size as the ranking criterion. (Other results have been left out due to space limitations.) There are 1000 peers and 1000 files. Each file may be annotated with an average of 30 terms from a dictionary of 5000 terms. Queries also draw terms from the dictionary, and have an average length of approximately two. (The exact length distribution follows the empirical distribution reported in [22].) The likelihood that a peer is reachable is 0.5. These values conform to the patterns observed in [23, 24].

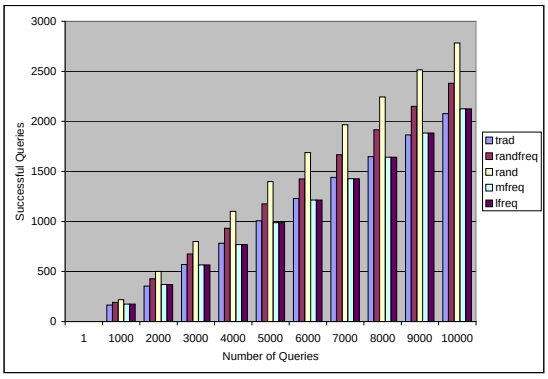

Fig. 1. Number of Successful Queries vs. Number of Queries.

The results shown in Figure 1 results are averages over five trials. They indicate that the **random** distribution of metadata outperforms all others by a significant margin. **Random** effectively increases the variety of metadata sets, and, due to the matching criterion, skews term frequecies toward terms that appear in queries. **Lfreq** and **mfreq** have a marginal effect, because they annotate

new replicas with either terms that seldom occur in queries (**lfreq**) or with terms that do not distinguish a file (**mfreq**). **Trad** fails because it does not increase the variety of metadata sets. **Randfreq** performs fairly, but, due to interaction with the matching criterion, overskews the overall distribution of terms for a file.

7 Conclusion

In this paper, we analyzed various methods of finding data in a P2P network: a P2P file sharing system; a Web search engine; and research systems in P2P IR. For one reason or another, each has a weakness due to its naive design, or the centralization of query processing or other components. These weaknesses limit their applicability in truly general, dynamic environments.

In response, we describe PIRS, which does not exhibit the weaknesses suggested above. Its functionality is centralized in the application level of the client, making few assumptions about the network layer or the behavior of its peers. It works by judiciously managing and using metadata that annotate the results of queries. This design is arguably more practical.

The question is whether it yields acceptable performance in terms of result quality. We showed by simulation that PIRS can improve the success of queries by over 30% compared with an ordinary P2P file sharing system, such as Gnutella.

7.1 The Outlook for Peer-to-Peer Information Retrieval

Industry trends seem to indicate that P2P information retrieval will be a strategic technology in the near future. Google is currently working on Puffin, a desktop search tool that helps users find information stored on their desktops [25]. Whether or not this is a counterattack to Microsoft's Longhorn [26] strategy, it signals a new focus on harnessing the information stored on desktops.

P2P file sharing has been a consistently active Internet activity for the last several years. This condition shows no sign of weakening, despite recent legal actions by the recording industries [27]. As the user base and variety of P2P applications grows, PIRS and other P2P search tools will only gain in significance.

References

1. Milojicic, D.S., Kalogeraki, V., Lukose, R., Nagaraja, K., Pruyne, J., Richard, B.: Peer-to-peer computing. Technical Report HPL-2002-57, Hewlitt-Packard Laboratories, Palo Alto (2002)
2. Noguchi, Y.: Online search engines help lift cover of privacy. Washington Post (2004) Feb. 9, 2004.
3. Hansell, S.: Yahoo to charge for guaranteeing a spot on its index. New York Times (2004) Mar. 2, 2004.
4. Cohen, B.: Bittorrent home page. (Web Document) bitconjurer.org/BitTorrent.
5. Watts, D.J., Strogatz, S.H.: Collective dynamics of small-world networks. Nature **393** (1998)

6. Tang, C., Xu, Z., Dwarkadas, S.: Peer-to-peer information retrieval using self-organizing semantic overlay networks. In: Proc. ACM SIGCOMM. (2003)
7. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S.: A scalable content-addressable network. In: Proc. ACM SIGCOMM. (2001)
8. LimeWire, LLC: Gnutella protocol 0.4. Web Document (2004) www9.limewire.com/developer/gnutella_protocol_0.4.pdf.
9. Lu, J., Callan, J.: Content-based retrieval in hybrid peer-to-peer networks. In: Proc. ACM Conf. on Information and Knowledge Mgt. (CIKM). (2003) 199–206
10. Nejdl, W., Wolf, B., Qu, C., Decker, S., Sintek, M., Naeve, A., Nilsson, M., Palmr, M., Risch, T.: Edutella: A p2p networking infrastructure based on rdf. In: Proc. World Wide Web Conf. (2002)
11. Ng, W., Ooi, B.C., Tan, K., Zhou, A.: Peerdb: A p2p-based system for distributed data sharing. In: Proc. IEEE Intl. Conf. on Data Eng. (ICDE). (2003)
12. Google, I.: Simplicity and enterprise search. Technical report, Google, Inc. (2003)
13. Ritter, J.: Why gnutella can't scale. no, really. Web Document (2001) www.darkridge.com/∼jpr5/doc/gnutella.html.
14. Singla, A., Rohrs, C.: Ultrapeers: Another step towards gnutella scalability. Technical report, Limewire, LLC (2002) rfc-gnutella.sourceforge.net/src/Ultrapeers_1.0.html.
15. Thadani, S.: Meta information searches on the gnutella network. (Web document) http://www.bearguru.com/kb/articles/metainfo_searches.htm.
16. Nilsson, M.: Id3v2 web site. Web Document (2004) www.id3.org.
17. Rohrs, C.: Search result grouping {in gnutella}. Technical report, LimeWire (2001) http://www.limewire.org/project/www/result_grouping.htm.
18. Free Peers, Inc.: Bearshare technical faq. Web document (2004) www.bearshare.com/help/faqtechnical.htm.
19. Grossman, D., Frieder, O.: Information Retrieval: Algorithms and Heuristics. Number ISBN 0-7923-8271-4. Kluwer Academic Publishers (1998)
20. Brin, S., Page, L.: The anatomy of a large scale hypertextual web search engine. In: Proc. World Wide Web Conf. (1998)
21. Schlosser, M.T., Condie, T.E., Kamvar, S.D.: Simulating a file-sharing p2p network. In: Proc. Wkshp. Semantics in Peer-to-Peer and Grid Comp. (2003)
22. Reynolds, P., Vahdat, A.: Efficient peer-to-peer keyword searching. In: Proc. ACM Conf. Middlware. (2003)
23. Ripeanu, M., Foster, I.: 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 (2002)
24. Saroiu, S., Gummadi, P.K., Gribble, S.D.: A measurement study of peer-to-peer file sharing systems. In: Proc. Multimedia Computing and Networking (MMCN). (2002)
25. Markoff, J.: Google moves toward clash with microsoft. New York Times (2004) May 19.
26. Microsoft, Inc.: Longhorn development center. Web Document (2004) msdn.microsoft.com/longhorn/.
27. Reardon, M.: Oops! they're swapping again. CNET News (2004)