Text-Based Content Search and Retrieval in *ad hoc* P2P Communities*

Francisco Matias Cuenca-Acuna and Thu D. Nguyen
{mcuenca, tdnguyen}@cs.rutgers.edu

Technical Report DCS-TR-483
Department of Computer Science, Rutgers University
110 Frelinghuysen Rd, Piscataway, NJ 08854

April 4, 2002

Abstract

We consider the problem of content search and retrieval in peer-to-peer (P2P) communities. P2P computing is a potentially powerful model for information sharing between ad hoc *groups of users because of its low cost of entry and natural model for resource scaling with community size. As P2P communities grow in size, however, locating information distributed across the large number of peers becomes problematic. We present a distributed text-based content search and retrieval algorithm to address this problem. Our algorithm is based on a state-of-the-art text-based document ranking algorithm: the vector-space model, instantiated with the TFxIDF ranking rule. A naive application of TFxIDF would require each peer in a community to collect an inverted index of the entire community. This is costly both in terms of bandwidth and storage. Instead, we show how TFxIDF can be approximated given compact summaries of peers' local inverted indexes. We make three contributions: (a) we show how the TFxIDF rule can be adapted to use the index summaries, (b) we provide a heuristic for adaptively determining the set of peers that should be contacted for a query, and (c) we show that our algorithm tracks TFxIDF's performance very closely, regardless of how documents are distributed throughout the community. Furthermore, our algorithm preserves the main flavor of TFxIDF by retrieving close to the same set of documents for any given query.*

1 Introduction

We consider the problem of content search and retrieval in peer-to-peer (P2P) communities. In the P2P computing model, each member in a community can contribute resources to the community and can establish direct connections with any other member to access communal resources or to carry out some communal activity [20]. P2P computing is a potentially powerful model for information sharing between *ad hoc* group of users because of its low cost of entry and explicit model for resource scaling with community size: any two users wishing to interact can form a P2P community. As individuals join the community, they will bring resources with them, allowing the community to grow naturally. P2P systems can scale to very large sizes if enough members join the community [17]. Measurements of one such community at Rutgers—a group of students have set up a file-sharing community connected by the dorm network—show over 500 users sharing over 6TB of data.

A number of open problems must be addressed, however, before the potential of P2P computing can be realized. Content search and retrieval is one such open problem. Currently, existing communities employ either centralized directory servers [18] or various flooding algorithms [11, 5, 29] for object location when given a name. Neither provides a viable framework for content search and retrieval. On the one hand, a centralized server presents a single point of failure and limits scalability. On the other hand, while flooding techniques can in theory allow for arbitrary content searches [19], in practice, typically only a name search, perhaps together with a limited number of attributes, is performed. Furthermore, flooding can be very expensive yet provide only limited power for locating relevant information (because queries can only be flooded to a portion of the community with little information to guide pruning). These techniques currently rely on heavy replication of popular items for successful searches. More recent works studying how to scale P2P communities have put forth more efficient and reliable distributed methods for name-based object location [17, 26, 23]. The focus, however, has remained on name-based object location because these efforts were intended to support P2P file systems, where there is a natural model for acquiring names.

*This work was supported in part by NSF grants EIA-0103722 and EIA-9986046.

As the amount of storage per person/device is rapidly growing, however, information management is becoming more difficult under the traditional file system hierarchical name space [10]. The success of Internet search engines is strong evidence that content search and retrieval is an intuitive paradigm that users can leverage to manage and access large volumes of information. As anecdotal evidence, a number of individuals in our research group use Google much more often than bookmarks, which provides a hierarchical namespace. These individuals even depend on Google to relocate pages that are used almost daily.

While P2P groups will not grow to the size of the web, with the exploding size and decreasing cost of storage, even small groups will share a large amount of data. Thus, we are motivated to explore a content search and retrieval engine that provides a similar information access paradigm to Internet search engines. In particular, we present a distributed text-based ranking algorithm for content search and retrieval in the specific context of PlanetP, an infrastructure that we are building to ease the task of developing P2P information-sharing applications.

Currently, PlanetP [6] provides a framework for *ad hoc* sets of users to easily set up P2P information sharing communities without requiring support from any centralized server[1]. PlanetP supports the indexing, searching and retrieval of information spread across a dynamic community of agents, possibly running on a set of heterogeneous devices. The basic idea in PlanetP is for each community member to create an inverted (word-to-document) index of the documents that it wishes to share, summarize this index in a compact form, and diffuse the summary throughout the community. Using these summaries, any member can query against and retrieve matching information from the collective information store of the community. (We provide an overview of PlanetP and discuss the advantages of its underlying approach for P2P information-sharing in Section 2.)

Thus, the problem that we focus on is how to perform text-based content search and retrieval using the index summaries that PlanetP uses. We have adopted a vector space ranking model, using the TFxIDF algorithm suggested by Salton et al. [24], because it is one of the currently most successful text-based ranking algorithm [28].

In a vector space ranking model, each document and query is abstractly represented as a vector, where each dimension is associated with a distinct word. The value of each component of the vector represents the importance of that word (typically referred to as the *weight* of the word) to that document or query. Given a query, we compute the relevance of a document to that query as some function of the angle between the two vectors.

TFxIDF is a popular method for assigning term weights under the vector space ranking model. The basic idea behind TFxIDF is that by using some combination of term frequency (TF) in a document with the inverse of how often that term shows up in documents in the collection (IDF), we can balance: (a) the fact that terms frequently used in a document are likely important to describe its meaning, and (b) terms that appear in many documents in a collection are not useful for differentiating between these documents for a particular query.

A naive application of TFxIDF would require each peer in a community to have access to the inverted index of the entire community (in order to compute the similarity between the query and all documents and provide a ranking). This is costly both in terms of bandwidth and storage. Instead, we show how TFxIDF can be approximated given PlanetP's compact summaries of peers' inverted indexes; we call the resulting algorithm TFxIPF. While we present TFxIPF in the specific context of PlanetP, it should be generally applicable to any framework that maintains some approximate information about the global index at each peer.

We make three contributions:

1. we show how the TFxIDF rule can be adapted to rank the peers in the order of their likelihood to have relevant documents, as well as rank the retrieved documents in the absence of complete global information;

2. we provide a heuristic for adaptively determining the set of peers that should be contacted for a query; and

3. using five benchmark collections from Smart [3] and TREC [14], we show that our algorithm matches TFxIDF's performance, despite the accuracy that it gives up by using a much more compact summary of the individual inverted indexes (rather than collecting the inverted index for the entire communal information store). Furthermore, our algorithm preserves the main flavor of TFxIDF, returning close to the same sets of documents for particular queries.

PlanetP trades some bandwidth for good search performance. Using our heuristics, PlanetP nearly matches the search performance (we will define performance metrics more precisely later in Section 4) of TFxIDF but, on average, will contact 20–40% more peers than if the entire inverted index was kept at each peer (40% only when we average over runs where we assume that users are willing to sort through a very large number of retrieved documents to find what he is looking for. Furthermore, the number of peers contacted under PlanetP is directed by a heuristic that can likely be further tuned for better performance.)

[1] We say "currently" because we are actively working to extend PlanetP to be a general framework for building P2P applications, not just information sharing.

Finally, we address our decision to explore/implement a text-based content search and retrieval algorithm. One might ask, aren't most P2P communities currently sharing non-text files such as music and movies? Yes. However, this is not the only information that could or should be shared by P2P communities. One can imagine companies forming P2P communities to share databases of (largely) text documents such as repositories of scientific papers, legal documents, etc. One can imagine a fantasy sport league community that shares a large collection of player and team information and statistics. More concretely, our research group is one information sharing community that would benefit from PlanetP; we currently use a centralized approach to share a large repository of research documents. This approach is not very satisfying for various reasons that we will not elaborate here but that we believe can be at least partially addressed under a decentralized control scheme. Even for non-text data (which we currently do not know how to search), there is often associated text data such as movie reviews that can be used effectively for text-based content search and retrieval.

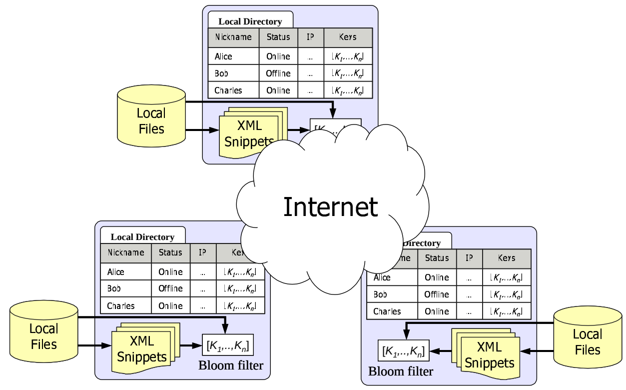


Figure 1. *A PlanetP community is comprised of a dynamic set of peers connected by a network such as the Internet. Each peer has a local store of XML documents that it wishes to share with the community. Peers use a gossiping algorithm to help each other maintain a directory of currently active peers as well as a set of Bloom filters summarizing the content of peers' local stores.*

2 PlanetP: Overview

As already mentioned, PlanetP supports the indexing, searching and retrieval of information spread across a dynamic community of agents (peers), possibly running on a set of heterogeneous devices. This section briefly discusses relevant features and design/implementation details of PlanetP to provide context for the rest of the paper. Figure 1 summarizes the components of PlanetP.

The basic data block in PlanetP is an XML snippet. These snippets contain text, from which we extract terms to be indexed[2], and possibly links (XPointers) to external files. To share an XML document, the user publishes the document to PlanetP, which indexes the document and stores a copy of it in a local data store. To share a non-XML document, the user publishes an XML snippet that contains a pointer to the file and possibly additional description of the file. PlanetP indexes the XML snippet and the external file if it is of a known type (e.g., PDF, Postscript, text, etc.). Also, PlanetP stores the XML snippet in the local data store but not the external file itself.

PlanetP uses a Bloom filter [1] to summarize the index of each peer. Briefly, a Bloom filter is an array of bits used to represent some set A—in this case, A is the set of words in the peer's inverted index. The filter is computed by obtaining n indices for each member of A, typically via n different hashing functions, and setting the bit at each index to 1. Then, given a Bloom filter, we can ask, is some element x a member of A by computing n indices for x and checking whether those bits are 1.

Once a peer has computed its Bloom filter, it diffuses it throughout the community by using a gossiping algorithm [6] that is a combination of Harchol-Balter et al.'s name dropper algorithm [13] and Demers et al.'s rumor mongering algorithm [7]. (This algorithm is also used to maintain a directory of peers currently on-line.) Each peer can then query for communal content by querying against the Bloom filters that it has collected. For example, a peer m can look for all documents containing the word *car*. It does this by testing for *car* in each of the Bloom filter. Suppose that this results in "hits" in the Bloom filters of peers $p1$ and $p2$. m then contacts $p1$ and $p2$ to see whether they indeed have documents containing the word *car*; note that these peers may not have any such documents since a Bloom filter can give *false positives*. On the other hand, this set of peers is guaranteed to be complete—that is, it is guaranteed that no peer other than $p1$ and $p2$ can have a document containing the word car—because Bloom filters can never give *false negatives*.

Our approach of diffusing index-summaries using Bloom filters has a number of advantages, the most significant of which are:

- As shall be seen, the collection of Bloom filters allows each peer to efficiently approximate the inverted

[2] Currently, we do not make use of the structure provided by XML tags. We plan to extend PlanetP to make use of this structure in the near future.

index of the *entire community*. This allows us to implement a ranking algorithm that depends on having global knowledge, which is either not possible or likely to be much more costly under current P2P systems [26, 23, 17].

- The Bloom filter is an efficient summary mechanism, minimizing the required bandwidth and storage at each node. On appendix A we show that PlanetP needs only 1% of the total data indexed to summarize the community's content.

- Peers can independently trade-off accuracy for storage. For example, a peer a may choose to combine the filters of several peers to save space; the trade-off is that a must now contact this set of peers whenever a query hits on this combined filter. This ability for independently trading accuracy for storage is particularly useful for peers running on memory-constrained devices (e.g., hand-held devices).

- Since the gossiped information is maintained at every peer in the community, PlanetP is extremely resistant to denial-of-service attacks.

The main disadvantage of using a diffusion approach is that new or rapidly changing information spreads slowly, as diffusion is necessarily spread out over time to minimize spikes in communication bandwidth. To address this problem, peers in PlanetP also implement an information brokerage service that uses consistent hashing [15] to publish and locate information. This second indexing service supports the timely location of new information, as well as the exchange of information between subsets of peers without involving the entire community (this is similar to the approach taken by Stoica et al. [26]). We will not discuss this feature further, however, since it does not impact the work we will present in this paper.

Using simulation, we have shown that PlanetP can easily scale to community sizes of several thousands. For example, using a gossiping rate of once per second[3], PlanetP can propagate a Bloom filter containing 1000 terms in less than 40 seconds for a community with 1000 peers. This spread of information requires an average of 24KB/s per peer. For communities connected by low bandwidth links, we can reduce the gossiping rate: reducing the gossiping rate to once every 30 seconds would require 9 minutes to diffuse a new Bloom filter, requiring an average of 2KB/s bandwidth. We have validated our simulation for communities of up to 200 peers on a cluster of PCs connected by a 100 Mb/s Ethernet LAN.

[3]When there is no new information to gossip, PlanetP dynamically reduces this gossiping rate over time to once-per-minute.

3 Distributed Content Search and Retrieval in PlanetP

The main problem that we are addressing in this paper is how to search for and retrieve documents relevant to a query posed by some member of a PlanetP community. Given a collection of text documents, the problem of retrieving the subset that is relevant to a particular query has been studied extensively (e.g., [24, 22]). Currently, one of the most successful techniques for addressing this problem is the vector space ranking model [24]. Thus, we decided to adapt this technique for use in PlanetP. In this section, we first briefly provide some background on vector space based document ranking, then we present our heuristics to adapt this technique to PlanetP's environment.

3.1 Vector Space Ranking

In a vector space ranking model, each document and query is abstractly represented as a vector, where each dimension is associated with a distinct term (word); the space would have k dimensions if there were k possible distinct terms. The value of each component of the vector represents the importance of that word (typically referred to as the *weight* of the word) to that document or query. Then, given a query, we rank the relevance of documents to that query by measuring the similarity between the query's vector and each of the candidate document's vector. The similarity between two vectors is generally measured as the cosine of the angle between them, computable using the following equation:

$$Sim(Q,D) = \frac{\sum_{t \epsilon Q} w_{Q,t} \times w_{D,t}}{\sqrt{|Q| \times |D|}} \quad (1)$$

where $w_{Q,t}$ represents the weight of term t for query Q and $w_{D,t}$ the weight of term t for document D. Observe that $Sim(Q,D) = 0$ means that D does not have any term that is in Q. A $Sim(Q,D) = 1$, on the other hand, means that D has every term that is in Q. Typically, the denominator of equation 1 is dropped for simplicity.

A popular method for assigning term weights is called the TFxIDF rule. The basic idea behind TFxIDF is that by using some combination of term frequency (TF) in a document with the inverse of how often that term shows up in documents in the collection (IDF), we can balance: (a) the fact that terms frequently used in a document are likely important to describe its meaning, and (b) terms that appear in many documents in a collection are not useful for differentiating between these documents for a particular query. For example, if we look at a collection of papers published in an Operating Systems conference, we will find that the terms *Operating System* appears in every document and therefore cannot be used to differentiate between the relevance of these documents.

Existing literature includes several ways of implementing the TFxIDF rule [24]. In our work, we adopt the following system of equations as suggested by Witten et al. [28]:

$$IDF_t = \log(1 + N/f_t)$$

$$w_{D,t} = 1 + \log(f_{D,t})$$

$$w_{Q,t} = IDF_t$$

where N is the number of documents in the collection, f_t is the number of times that term t appears in the collection, and $f_{D,t}$ is the number of times term t appears in document D.

If we simply drop the normalizing denominator from equation 1, then long documents will be erroneously ranked higher than short documents because they have higher term weights (because of higher term frequencies). Therefore it is customary to substitute this simpler normalization factor $|D| = \sqrt{\text{number of terms in document D}}$. The resulting similarity measure is

$$Sim(Q, D) = \frac{\sum_{t \epsilon Q} w_{D,t} \times IDF_t}{|D|} \tag{2}$$

Given a collection of documents, current search engines implement this ranking algorithm by constructing an inverted index over the collection [28]. This index associates a list of documents with each term, the weight of the term for each document, and the positions where the terms appear. Further, information like the inverse document frequency (IDF) and other useful statistics are also added to the index to speed up query processing. An engine can then use this inverted index to quickly determine the subset of documents that contain one or more terms in some query Q, and to compute the vectors needed for equation 2. Then, the engine can rank the documents according to their similarity to the query and present the results to the user.

3.2 Indexing and Searching in PlanetP

Recall that in PlanetP, a member of the community only distributes a summary of its inverted index using a Bloom filter, not the inverted index itself. Thus, we cannot use the above TFxIDF rule directly and so the challenge is to adapt this algorithm to the information available at each peer in PlanetP. Our adapted algorithm works as follows. We break the relevance ranking problem into two sub-problems: (1) ranking peers according to the likelihood of each peer having documents relevant to the query, and (2) deciding on the number of peers to contact and ranking the documents returned by these peers.

The node ranking problem. We apply the same idea behind TFxIDF to rank peers. In particular, we introduce a measure called the *inverse peer frequency* (IPF). For a term t, IPF_t is computed as $\log(1 + N/N_t)$, where N is number of peers in the community and N_t is the number of peers that have one or more documents with term t in it. Similar to IDF, the idea behind this metric is that a term that is present in the index of every peer is not useful for differentiating between the peers for a particular query. Note that IPF can conveniently be computed using the Bloom filters collected at each peer: N is the number of Bloom filters, N_t is the number of hits for term t against these Bloom filters.

Given the above definition of IPF, we then propose the following relevance measure for ranking peers:

$$R_i(Q) = \sum_{t \epsilon Q \wedge t \epsilon BF_i} IPF_t \tag{3}$$

which is simply a weighted sum over all terms in the query of whether a peer contains that term, weighted by how useful that term is to differentiate between peers. t is a term, Q is the query, and BF_i is the set of terms represented by the Bloom filter of peer i, and R_i is the resulting relevance of peer i to query Q. Intuitively, this scheme gives peers that contain all terms in a query the highest ranking. Peers that contain different subsets of terms are ranked according to the power of these terms for differentiating between peers with potentially relevant documents.

The selection problem. As communities grow in size, it is neither feasible nor desirable to contact a large subset of peers for each query. Thus, once we have established a relevance ordering of peers for a query, we must then decide how many of them to contact. To address this problem, we first assume that the user specifies an upper limit k on the number of documents that should be returned in response to a query. Then, a simple solution to the selection problem would be to contact the peers one by one, in the order of their relevance ranking, until we have retrieved k documents.

As shall be seen in Section 4, however, this obvious approach leads to terrible performance as measured by the percentage of relevant documents returned. The reason behind this poor performance is that, when a peer is contacted, it may return say m documents. In most cases, not all m returned documents are highly relevant to the query. Thus, by stopping immediately once we have retrieved k documents, a large subset of the retrieved documents may have very little relevance to the query. To illustrate this problem more concretely, let us assume that there are 5 candidate peers and that the user is willing to accept up to 10 documents. Each of the peers stores 2 highly ranked documents and 8

Trace	Queries	Documents	Number of words	Collection size (MBs)
CACM	52	3204	75493	2.1
MED	30	1033	83451	1.0
CRAN	152	1400	117718	1.6
CISI	76	1460	84957	2.4
AP89	97	84678	129603	266.0

Table 1. *Characteristic of the collections used to evaluate PlanetP search and retrieval capabilities.*

documents with low rankings. If we contact all 5 peers, we will collect 50 documents, and, hopefully after ranking them, we will return to the user only the 10 highly ranked ones. On the other hand, if we allow PlanetP to stop searching as soon as 10 related documents have been obtained, it will only contact one node and it will return only 2 highly ranked documents and 8 that are of low relevance.

To address this problem, we introduce the following heuristic for adaptively determining a stopping point. Given a relevance ordering of peers, contact them one-by-one from top to bottom. Maintain a relevance ordering of the documents returned using

$$Sim(Q,D) = \frac{\sum_{t \epsilon Q} w_{D,t} \times IPF_t}{|D|} \qquad (4)$$

Stop contacting peers when the documents returned by a sequence of p peers fail to contribute to the top k ranked documents. Intuitively, the idea is to get an initial set of k documents and then keep contacting nodes only if the chance of them being able to provide documents that contribute to the top k is relatively high. Using experimental results from a number of known document collections (see Section 4), we propose the following function for p

$$p = \left\lfloor 2 + \frac{N}{300} \right\rfloor + 2 \left\lfloor \frac{k}{50} \right\rfloor \qquad (5)$$

where N is the size of the community.

This linear function on community size and k gives a p that ranges between 2 and 9 for communities of up to 1000 nodes and k of up to 140 documents.

4 Evaluating PlanetP's Search Heuristics

We now turn to assessing the performance of TFxIPF together with our adaptive stopping heuristic as implemented in PlanetP. We measure performance using two accepted metrics, *recall* (R) and *precision* (P), which are defined as follows:

$$R(Q) = \frac{\text{no. relevant docs. presented to the user}}{\text{total no. relevant docs. in collection}} \qquad (6)$$

$$P(Q) = \frac{\text{no. relevant docs. presented to the user}}{\text{total no. docs. presented to the user}} \qquad (7)$$

where Q is the query posted by the user. $R(Q)$ captures the fraction of relevant documents a search and retrieval algorithm is able identify and present to the user. $P(Q)$ describes how much irrelevant material the user may have to look through to find the relevant material.

Ideally, one would like to retrieve all the relevant documents and not a single irrelevant one. If we did this, we would obtain a 100% recall and 100% precision. Unfortunately, none of the current ranking schemes is able to achieve this performance when assessed against a number of benchmark collections and queries, where humans have manually determined the set of documents relevant to each query. Current algorithms provide a tradeoff between R and P: in order to find more relevant documents, the user must be willing to look through increasing amount of irrelevant material.

We assess the performance of PlanetP by comparing its achieved recall and precision against the original TFxIDF algorithm. If we can match the TFxIDF's performance, then we can be confident that PlanetP provides state-of-the-art search and retrieval capabilities[4], despite the accuracy that it gives up by gossiping Bloom filters rather than the entire inverted index.

Finally, in addition to recall and precision, we also examine the average number of peers that must be contacted per query under PlanetP. Ideally, we would want to contact as few peers as possible to minimize resource usage per query. We study the number of peers that must be contacted as a function of the number of documents the user is willing to view and the size of the community.

[4]when only using the textual content of documents, as compared to link analysis as is done by Google and other web search engines [2]

Collection	Query
AP89	<top> <head> Tipster Topic Description <num> Number: 065 <dom> Domain: Science and Technology <title> Topic: Information Retrieval Systems <desc> Description: Document will identify a type of information retrieval system. <smry> Summary: A relevant document will identify a new information retrieval system, identify the company or person marketing the system, and identify some of the characteristics of the system. <narr> Narrative: A relevant document will identify an information retrieval system, identify the company or person marketing the system, and identify some of the characteristics of the system. <con> Concept(s): 1. information retrieval system 2. storage, database, data, query <fac> Factor(s): <def> Definition(s): </top>
MED	.I 8 .W the effects of drugs on the bone marrow of man and animals, specifically the effect of pesticides. also, the significance of bone marrow changes.

Table 2. *Sample queries for the AP89 and MED collections. Note that for the AP89 query, we only use the keywords in the* <con> *section.*

4.1 Experimental Environment

We use five collections of documents (and associated queries and human relevance ranking) to measure PlanetP's performance; Table 1 presents the main characteristics of these collections. Four of the collections, CACM, MED, CRAN, and CISI were previously collected and used by Buckley to evaluate Smart [3]. These collections are comprised of small fragments of text and summaries and so are relatively small in size. The last collection, AP89, was extracted from the TREC collection [14] and includes full articles from Associated Press published in 1989. Table 2 shows an example query from each of the MED and AP89 collections.

To measure PlanetP's recall and precision on the above collections, we built a simulator that first distributes documents across a set of virtual peers and then runs and evaluates different search and retrieval algorithms. To compare PlanetP with TFxIDF, we assume the following optimistic implementation of TFxIDF: each peer in the community has the full inverted index and word count needed to run TFxIDF using ranking equation 2. For each query, TFxIDF would compute the top k ranking documents and then contact the exact peers required to retrieve these documents. In both cases, TFxIDF and TFxIPF, the simulator will pre-process the traces by doing stop word removal and stemming. The former tries to eliminate frequently used words like "the", "of", etc. and the second tries to conflate words to their root (e.g. "running" becomes "run").

We study PlanetP's performance under two different distributions of documents among peers in the community: (a) uniform, and (b) Weibull. The motivation for studying the uniform distribution of documents among a set of peers is that it presents the worst case for a distributed search and retrieval algorithm. The documents relevant to a query are likely spread across a large number of peers. The distributed search algorithm must find all these peers and contact them.

The motivation for studying a Weibull distribution arises from measurements of current P2P file-sharing communities. For example, Saroiu et al. found that 7% of the users in the Gnutella community share more files than all the rest together [25]. We have also studied a community that may be representative of future communities based on PlanetP. In particular, students with access to the Rutgers's dormitory network have created a file-sharing community comprised of more than 500 users, sharing more than 6TB of data.

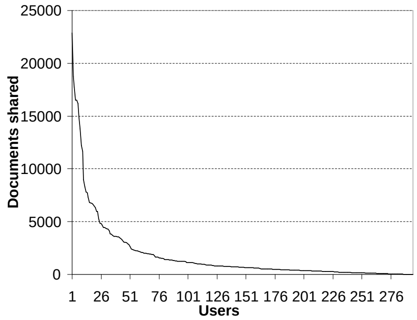


Figure 2. *Number of files shared by each user on the Rutgers dorm network. The users are sorted according to the amount of files they share.*

They use a software package called Direct Connect [8] that resembles an IRC chat channel. The Direct Connect hub maintains a list of online users and uses a robot to crawl each user's directory structure. Information about the files shared is maintained at the hub and can be queried by the users.

Studying this community, we observed a data distribution that is very similar to that found by Saroiu et al. In particular, we found that around 9% of the users are responsible for providing the majority of the files in the community. Figure 2 shows the number of files shared by every user observed during one global snapshot of the community. We fitted this data to a Weibull distribution and used this theoretical distribution with the extracted parameters ($\alpha = 0.7$, $\beta = 46$) to drive the partitioning of a collection among a simulated community.

Finally, unless noted otherwise, we will use the Smart collections on communities of 100 nodes and the AP89 collection on communities of 400 nodes to account for the difference in their sizes.

4.2 Search and Retrieval

To evaluate PlanetP's search and retrieval performance, we assume that when posting a query, the user also provides the parameter k, which is the maximum number of documents that he is willing to accept in answer to a query. Figure 3 plots TFxIDF's and PlanetP's average recall and precision over all provided queries as functions of k for the MED and AP89 collections, respectively. We only show results for the MED collection instead of all four Smart collections to save space. Results for the MED collection is representative of all four. We refer the reader to our web site, http://www.panic-lab.rutgers.edu/, for results for all collections.

We make several observations. First, using TFxIPF and our adaptive stopping condition, PlanetP tracks the performance of TFxIDF closely. For the AP89 collection, PlanetP performs slightly worse than TFxIDF for $k < 150$ but catches up for larger k's. For the MED collection, PlanetP gives nearly identical recall and precision to TFxIDF. In fact, at large k, TFxIPF slightly outperforms TFxIDF. While the performance difference is negligible, it is interesting to consider how TFxIPF can outperform TFxIDF; this is possible since TFxIDF is not always correct. In this case, TFxIPF is finding lower ranked documents that were determined to be relevant to queries, while some of the highly ranked documents returned by TFxIDF, but not TFxIPF, were not relevant.

Second, PlanetP's adaptive stopping heuristic is critical to performance. If we simply stopped retrieving documents as soon as we have gotten k documents, recall and precision would be much worse than TFxIDF, as shown by the IPF First-k curves. Finally, as expected, as k increases, recall improves at the expense of precision, although for both collections, precision was still relatively high for large k's (e.g., at $k = 40$, precision is about 40% and recall is about 60% for the MED collection.)

Figure 3 plotted the performance of PlanetP against k for a single community size: 100 peers for MED and 400 peers for AP89. In Figure 4, we plot the recall when k is 20 against community size to study PlanetP's scalability. We only show results for the AP89 collection as the others were too small to accommodate a wide range of community sizes. We show the performance of TFxIPF with two variants of the stopping heuristic: one that is a function of both k and N, the number of peers, and one that is just a function of k.

We make two observations. First, PlanetP's recall remains constant even when the community size changes by an order of magnitude, from 100 to 1000 peers. Second, the fact that our adaptive stopping heuristic is a function of both k and community size is critical. When the adaptive stopping heuristic only accounts for varying k, recall degrades as community size grows. This is because the relevant documents become spread out more thinly among peers as the community size increase. Thus, the stopping heuristic should allow PlanetP to widen its search by contacting more peers.

4.3 Number of Peers Contacted

To better understand the effects of our adaptive stopping heuristic, we present in Figure 5 the number of nodes contacted when using TFxIDF and all variants of TFxIPF as well as the lower bound on the number of nodes that need to be contacted. To compute the lower bound, we sort the nodes according to the number of relevant documents they

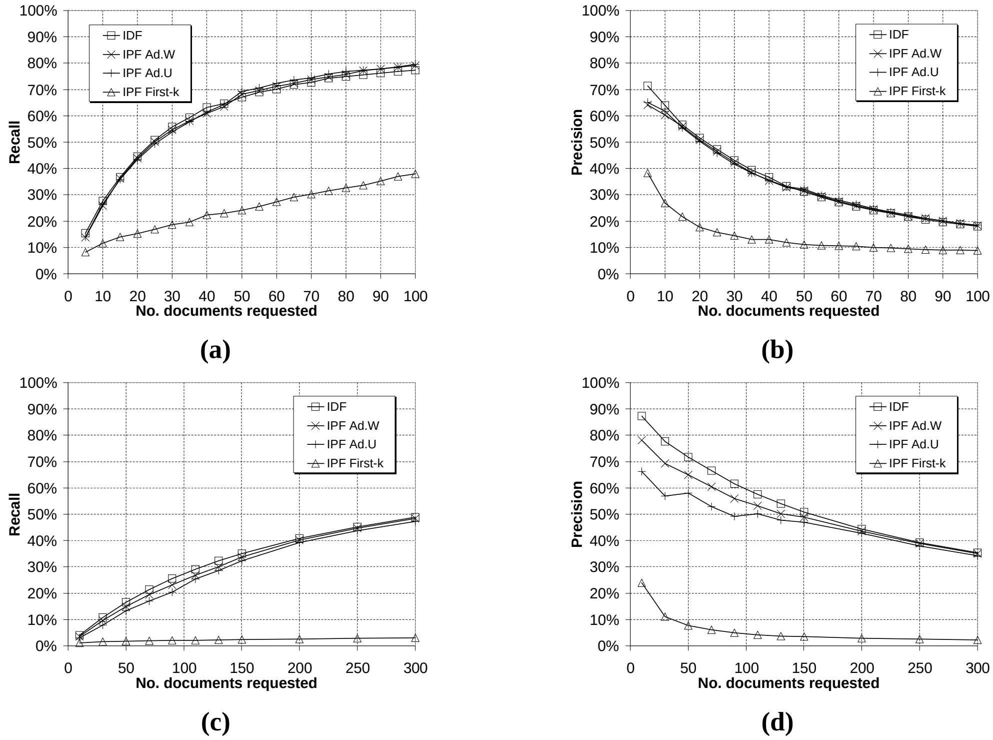


Figure 3. *Average (a) recall and (b) precision for the MED collection distributed among 100 peers. Average (c) recall and (d) precision for the AP89 collection distributed among 400 peers. IDF is TFxIDF. IPF Ad.W is TFxIPF with the adaptive stopping heuristic on the Weibull distribution of documents. IPF Ad.U is TFxIPF with the adaptive stopping heuristic on the uniform distribution of documents. IPF First-k is TFxIPF that stops immediately after first k documents have been retrieved.*

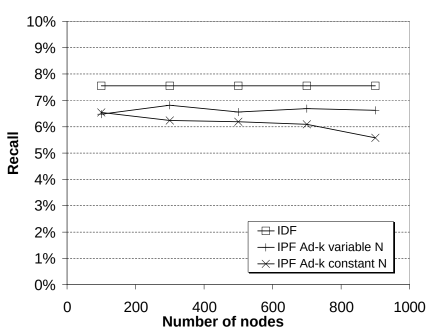
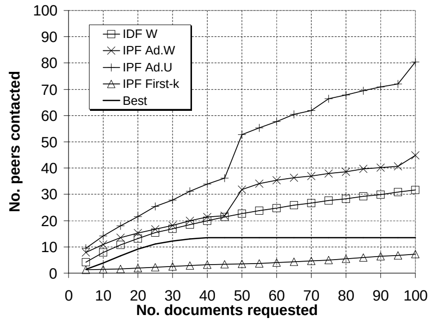


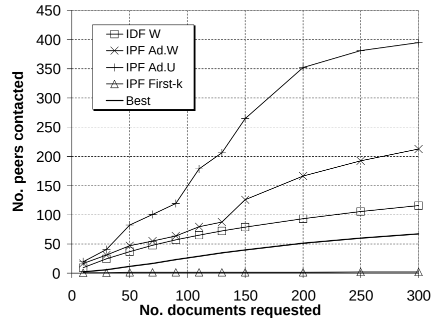
Figure 4. *Number of relevant documents retrieved as a function of community size when k is kept constant at 20. IPF Ad-k variable N is TFxIPF with the adaptive stopping heuristic. IPF Ad-k constant N is TFxIPF with a stopping heuristic that is only a function of k and not of community size. All these plots where obtained using a Weibull distribution of documents on the AP89 collection.*

store (assuming global knowledge of the human ranking) and then we plot the lowest number of nodes needed to get k relevant documents (for 100% precision). Note that the lower bound is different than the number of peers contacted by TFxIDF because it is based on the provided human relevance measure (which is binary), not the TFxIDF ranking.

Again, we make several observations. First, our adaptive stopping heuristic is critical for increasing recall with increasing k because it causes more nodes to be contacted. In fact, to match TFxIDF's performance, PlanetP has to contact more peers than TFxIDF at large k's. This is because PlanetP has less information than assumed for TFxIDF, and so may contact peers that don't have highly ranked documents. On the other hand, simply stopping as soon as we have retrieved k potentially relevant document gives very little growth in the number of peers contacted. As a result, it contacts many less peers than the lower bound imposed by the relevance judgments. This helps to explain the recall and precision for the various algorithms shown earlier. Second, beyond a certain k, 50 for MED and 150 for TREC, PlanetP starts to contact significantly more peers than TFx-



(a)



(b)

Figure 5. *Number of peers contacted when requesting different numbers of documents for (a) the MED collection distributed across 100 peers and (b) the AP89 collection distributed across 400 peers. IDF W is TFxIDF. IPF Ad.W is TFxIPF with the adaptive stopping heuristic. IPF Ad.U is TFxIPF with the adaptive stopping heuristic. IPF First-k is TFxIPF that stops immediately after first k documents have been retrieved. Best is the minimum number of nodes that can be contacted to retrieve k documents using the relevance judgments. All these plots use a Weibull distribution of documents except for IPF Ad.U, which uses a uniform distribution.*

IDF. At corresponding k, PlanetP's recall improves relative to TFxIDF: PlanetP outperforms TFxIDF slightly for MED and becomes essentially equal to TFxIDF. This implies that either equation 5 is too strongly dependent on k or that the relationship is not linear. We are currently working to refine our stopping heuristic to see whether we can reduce the number of peers contacted at large k without degrading performance too much. Third, PlanetP has to work much harder under the uniform distribution because relevant documents are spread out throughout the community. Thus, actual observations of Weibull-like distributions with shape parameters of 0.7 actually work in favor of a distributed search and retrieval engine such as PlanetP. Note that the results for PlanetP under the uniform distribution is not directly comparable to those for TFxIDF because we only studied TFxIDF under the Weibull distribution; we did not study TFxIDF under the uniform distribution because the distribution does not change TFxIDF's recall and precision; only the number of peers contacted. Finally, our adaptive stopping heuristic allows PlanetP to work well regardless of the distribution of relevant documents. It allows PlanetP to widen its search when documents are more spread out. It helps PlanetP to contract its search when the documents are more concentrated.

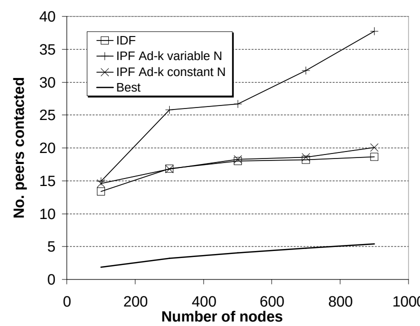


Figure 6. *Number of peers contacted as a function of the community size when k is kept constant at 20. IPF Ad-k variable N is TFxIPF with the adaptive stopping heuristic. IPF Ad-k constant N is TFxIPF with a stopping heuristic that is only a function of k and not of community size. All these plots where obtained using a Weibull distribution of documents on the AP89 collection.*

Finally, we study the effect of making our adaptive stopping heuristic a function of community size; Figure 6 plots the number of nodes contacted against community size for the AP89 collection for TFxIPF with an adaptive stopping heuristic that adapts to the community size and one that does not. Previously, we saw that adapting to community size was important to maintain a constant recall as community size increase. This figures shows the reason: if we do not adapt to community size, the stopping heuristic throttles the number of peers contacted too quickly. With increasing community size, the number of nodes contacted drops below that of TFxIDF, resulting in lower recall as previously shown.

k	Recall	Overlap
5	14%	68%
10	26%	69%
15	36%	78%
20	44%	79%

Table 3. *Amount of overlap on the documents returned by TFxIDF and TFxIPF when asking for different amounts of results (recall levels). This results where obtained on the MED collection.*

4.4 Does PlanetP Retrieve Similar Documents to TFxIDF?

We conclude our study of PlanetP's search and retrieval algorithm by considering whether the modified TFxIPF rule finds the same set of documents as TFxIDF. Table 3 gives the average intersection between the sets of relevant documents returned by TFxIDF and those returned by PlanetP in response to queries against the MED collection. We only show the intersections for low recall because at high recall, by definition, the intersection will approach 100%.

We observe that even at relatively low recall, 10–40%, the intersection is close to 70%, indicating that TFxIPF finds essentially the same set of relevant documents as TFxIDF. This gives us confidence that our adaptations did not change the essential ideas behind TFxIDF's ranking.

5 Related Work

Current P2P file-sharing applications such as Napster [18], Gnutella [11], and KaZaA [16] all support name-based document search and retrieval. While these systems have been tremendously successful, name-based search is frustratingly limited. These systems rely on heavy replication of popular items for successful searches, and communities built on them have been limited to sharing music and video files. Our goal for PlanetP is to increase the power with which users can locate information in P2P communities. Also, we have focused more tightly on text-based information, which is more appropriate for collections of scientific documents, legal documents, inventory databases, etc. These differences have led us to design and implement a text-based content search and retrieval engine, as presented in this paper.

In contrast to existing systems, recent research efforts in P2P seek to provide the illusion of having a global hash table shared by all members of the community. Frameworks like Tapestry [30], Pastry [23], Chord [26] and CAN [21] use different techniques to spread (key, value) pairs across the community and to route queries from any member to where the data is stored. Although this distributed hash structure could be used to create an inverted index, it would not be as efficient as the approach proposed by PlanetP. In all these systems, there is a time cost needed to contact the right node in order to publish a single key. If we want to share a document's content then we need to publish every unique word contained in it. Therefore the time needed will be almost linear on the document size. On the other hand, PlanetP can publish all the keys in the same Bloom filter with no need to contact every node since the information will be gossiped according to the bandwidth available. Furthermore, none of these frameworks provides the infrastructure needed to implement ranking (although PlanetP's ranking algorithm could be adapted for use in these systems).

More related to PlanetP's goals, Cori [4] and Gloss [12] address the problems of database selection and ranking fusion on distributed collections. Recent studies done by French et al. [9] show that both scale well to 900 nodes. Although they are based on different ranking techniques, the two rely on similar collection statistics. Gloss uses $Df_{i,t}$ (number of documents in node i with term t) and $SumW_{i,t}$ (sum of the local weights for term t at node i) while Cori needs only $Df_{i,t}$. In both cases the amount of information used to rank nodes is significantly smaller than having a global inverted index. Gloss needs only 2% of the space used by a global index. The paradigm used in Gloss and Cori is similar to having a server (or a hierarchy of servers) that will be available for users to decide which peers to contact. In PlanetP, we want to empower peers to work autonomously and therefore we distribute Bloom filters widely so they can answer queries even in the presence of network and node failures. In order to accommodate peers with scarce resources, we minimize the information shared and its maintenance cost. PlanetP does not propagate term frequency and can tolerate false positives on the communal data. These two characteristics allow us to use an efficient data representation that can trade space for false positives (performance). In appendix A we show that a peer needs to store around 1% of the total information shared.

Finally JXTA Search [27] provides mechanisms for clients to route queries to the right data sources. This is similar to the database selection problem, but they have implemented it using a publish/subscribe model. In JXTA Search, peers that want to provide content give a sketch of the queries that they are willing to answer to a hub. Using hubs, clients can find which nodes are suitable for a particular question. Unlike PlanetP, in their architecture, peers do not get any ranking of the results and JXTA Search does not try to contact the most relevant nodes for the query.

6 Conclusions

P2P computing is a potentially powerful model for information sharing between *ad hoc* communities of users. It allows users to leverage existing desktop computing power, as opposed to requiring specialized servers. Further, resources will adaptively scale with community size because new members contribute additional resources when they join a community.

As P2P communities grow in size, however, locating information distributed across the large number of peers becomes problematic. In this paper, we have presented a text-based ranking algorithm for content search and retrieval. Our thesis is that the search paradigm, where a small set of relevant terms is used to locate documents, is as natural as locating documents by name, as demonstrated by the success of the web search engines. To be useful, however, the search and retrieval algorithm must successfully locate the information the user is searching for, without presenting too much unrelated information.

We chose to adapt a well-known state-of-the-art text-based document ranking algorithm, the vector-space model, instantiated with the TFxIDF ranking rule. A naive application of TFxIDF would require each peer in a community to have access to the inverted index of the entire community. This is costly both in terms of bandwidth and storage. Instead, we show how TFxIDF can be approximated given a compact summary (the Bloom filter) of each peer's inverted index. We make three contributions: (a) we show how the TFxIDF rule can be adapted to use the index summaries, (b) we provide a heuristic for adaptively determining the set of peers that should be contacted for a query, and (c) we have shown that our algorithm tracks TFxIDF's performance very closely, regardless of how documents are distributed throughout the community. Finally, our algorithm preserves the main flavor of TFxIDF by returning much the same set of documents for a particular query.

A good content-based search and retrieval algorithm like ours will make it much easier for P2P communities to share large amounts of information.

References

[1] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[2] S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, 1998.

[3] C. Buckley. Implementation of the SMART information retrieval system. Technical Report TR85-686, Cornell University, 1985.

[4] J. P. Callan, Z. Lu, and W. B. Croft. Searching Distributed Collections with Inference Networks . In *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 21–28, 1995.

[5] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Workshop on Design Issues in Anonymity and Unobservability*, pages 46–66, 2000.

[6] F. M. Cuenca-Acuna, C. Peery, R. P. Martin, and T. D. Nguyen. PlanetP: Infrastructure Support for P2P Information Sharing. Technical Report DCS-TR-465, Department of Computer Science, Rutgers University, Nov. 2001.

[7] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database maintenance. In *Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing*, pages 1–12, 1987.

[8] Direct Connect. http://www.neo-modus.com.

[9] J. C. French, A. L. Powell, J. P. Callan, C. L. Viles, T. Emmitt, K. J. Prey, and Y. Mou. Comparing the performance of database selection algorithms. In *Research and Development in Information Retrieval*, pages 238–245, 1999.

[10] D. K. Gifford, P. Jouvelot, M. A. Sheldon, and J. W. O. Jr. Semantic File Systems. In *Proceedings of the 13th ACM Symposium on Operating Systems Principles*, 1991.

[11] Gnutella. http://gnutella.wego.com.

[12] L. Gravano, H. Garcia-Molina, and A. Tomasic. The effectiveness of gloss for the text database discovery problem. In *Proceedings of the ACM SIGMOD Conference*, pages 126–137, 1994.

[13] M. Harchol-Balter, F. T. Leighton, and D. Lewin. Resource discovery in distributed networks. In *Symposium on Principles of Distributed Computing*, pages 229–237, 1999.

[14] D. Harman. Overview of the first trec conference. In *Proceedings of the 16th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 1993.

[15] D. R. Karger, E. Lehman, F. T. Leighton, R. Panigrahy, M. S. Levine, and D. Lewin. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *ACM Symposium on Theory of Computing*, pages 654–663, 1997.

[16] KaZaA. http://www.kazaa.com/.

[17] J. Kubiatowicz, D. Bindel, Y. Chen, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. Oceanstore: An architecture for global-scale persistent storage. In *Proceedings of ACM ASPLOS*, 2000.

[18] Napster. http://www.napster.com.

[19] A. Oram, editor. *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*. O'Reilly Press, 2001.

[20] M. Parameswaran, A. Susarla, and A. B. Whinston. P2P Networking: An Information Sharing Alternative. *IEEE Computer*, 34(7), 2001.

[21] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content addressable network. In *Proceedings of the ACM SIGCOMM '01 Conference*, 2001.

[22] S. E. Robertson and K. S. Jones. Relevance weighting of search terms. In *Journal of the American Society for Information Science*, volume 27, pages 129–146, 1976.

[23] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, 2001.

[24] G. Salton, A. Wang, and C. Yang. A vector space model for information retrieval. In *Journal of the American Society for Information Science*, volume 18, pages 613–620, 1975.

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

[26] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the ACM SIGCOMM '01 Conference*, 2001.

[27] S. Waterhouse. Jxta search: Distributed search for distributed networks. Sun Microsystems, Inc., 2001.

[28] I. Witten, A. Moffat, and T. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufmann, San Francisco, second edition, 1999.

[29] B. Yang and H. Garcia-Molina. Efficient search in peer-to-peer networks. In *Proceedings of the International Conference on Distributed Computing Systems (ICDCS)*, July 2002.

[30] Y. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, University of California, Berkeley, 2000.

Appendix A - PlanetP's memory usage

In this appendix, we present how we estimated the amount of memory needed by each PlanetP's member to keep track of the community's content. Note that the memory usage depends mainly on the Bloom filter size and the number of peers on the community. In our calculation we have chosen Bloom filters that are able to store each peer's set of terms with less than 5% of false positives. For example, if we spread the AP89 collection across a community of 1000 peers, each peer will receive on average 4500 terms. On this scenario a 4.6KB filter will store a single peer's data, which means that the whole community can be summarized with 4.6MB of memory. Because nodes exchange filters in compressed form, the bandwidth required by a single node to gather the remaining 999 filters will be 3.3MB.

Table 4 shows the results obtained for different community sizes using the same calculations as presented above.

No. peers	Memory used (MB)	% of collection size
10	0.45	0.18%
100	1.79	0.70%
1000	4.48	1.76%

Table 4. *Amount of memory used per node to store Bloom filters summarizing the whole community on AP89.*