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

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

Department of Computer Science, Rutgers University
110 Frelinghuysen Rd, Piscataway, NJ 08854

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. As P2P communities grow, however, locating information distributed across the large number of peers becomes problematic. We address this problem by adapting a state-of-the-art text-based document ranking algorithm, the vector-space model instantiated with the TFx-IDF ranking rule, to the P2P environment. We make three contributions: (a) we show how to approximate TFxIDF using compact summaries of individual peers' inverted indexes rather than the inverted index of the entire communal store; (b) we develop 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, giving P2P communities a search and retrieval algorithm as good as that possible assuming a centralized server.

1 Introduction

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 explicit model for resource scaling: 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. Measurements of one such community at Rutgers show over 500 users sharing over 6TB of data. Open communities such as Gnutella [11] have achieved much greater sizes.

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 [16] or various flooding algorithms [11, 5, 26] 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 [17], in practice, typically only a name search, perhaps together with a limited number of attributes, is performed.

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

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 [15, 24, 21]. 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.

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. While P2P groups will not grow to the size of the web, with the exploding capacity 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]. 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. [22], because it is one of the currently most successful text-based ranking algorithm [25]. Under this model, a query is comprised of a set of terms. For each document in the collection, TFxIDF uses the frequency of each query term in that document and the frequency of the term across the collection to compute the likely relevance of the document to the query.

A naive application of TFxIDF would require each peer in a community to have access to the inverted index of the entire communal store. 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. (Note that while we present adaptation 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 say "currently" because we are actively working to extend PlanetP to be a general framework for building P2P applications, not just information sharing.

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 develop 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 [13], we show that our algorithm matches TFxIDF's performance, despite the accuracy that it gives up by using only summaries of the individual inverted indexes rather than the inverted index of the entire communal 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[2].

2 PlanetP: Overview

PlanetP is an infrastructure that we are building to support the indexing, searching and retrieval of information spread across a dynamic community of peers, possibly running on a set of heterogeneous devices [6]. This section briefly discusses relevant features and design/implementation details of PlanetP to provide context for the rest of the paper.

The basic data block in PlanetP is an XML snippet. These snippets contain text, from which we extract terms to be indexed[3], 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 using a gossiping algorithm [7, 6]. (This algorithm is also used to maintain a directory

[2] 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. Further, our stopping heuristic currently allows for overly aggressive growth in peers contacted as a function of both community size and of the number of documents to be returned. We are tuning this function to reduce the number of peers contacted without degrading TFxIPF's accuracy.

[3] 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.

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* by testing for *car* in each of the Bloom filters. 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: (1) The Bloom filter is an efficient summary mechanism, minimizing the required bandwidth and storage at each node. In appendix A, we show that PlanetP only needs approximately 1% of the total data indexed to summarize the community's content. (2) Previous studies of file systems have shown that a majority of files change very slowly [20, 8]. If P2P information collections display the same characteristic, then, using Bloom filters, PlanetP will place very little load on the community for searches against this bulk of slowly changing data. (3) 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). (4) A peer can know that documents relevant to a query might exist on peers that are currently off-line. Thus, instead of missing these documents as in current systems, the searching peer could arrange to rendezvous with the off-line peers when they reconnect to obtain the needed information.

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 [4], 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.

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., [22, 19]). Currently, one of the most successful techniques for addressing this problem is the vector space ranking model [22]. Thus, we decided to adapt this technique for use in PlanetP. In this section,

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

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 vectors. 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, $|Q|$ is dropped from the denominator of equation 1 since it is constant for all the documents.

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 [22]. In our work, we adopt the following system of equations as suggested by Witten et al. [25]:

$$IDF_t = \log(1 + N/f_t) \qquad w_{D,t} = 1 + \log(f_{D,t}) \qquad w_{Q,t} = IDF_t$$

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.

The resulting similarity measure is

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

where $|D|$ = the number of terms in document D.

Given a collection of documents, current search engines implement this ranking algorithm by constructing an inverted index over the collection [25]. 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 Search and Retrieval in PlanetP

We cannot implement the above relevance ranking directly in PlanetP because we do not have all the necessary information. Instead, we approximate this function by breaking the 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. To rank peers, 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. Unlike IDF, 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, 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 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 equation 2 with IPF_t substituted for IDF_t. 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

$$p = \left\lfloor 2 + \frac{N}{300} \right\rfloor + 2 \left\lfloor \frac{k}{50} \right\rfloor \tag{4}$$

where N is the size of the community.

Note that while we have presented the above algorithm as contacting peers one-by-one, to reduce query response time, we might choose to contact peers in groups of m peers at a time. Such a parallel algorithm trades off potentially contacting some peers unnecessarily for shorter response time.

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:

$$R(Q) = \frac{\text{no. relevant docs. presented to the user}}{\text{total no. relevant docs. in collection}} \tag{5}$$

$$P(Q) = \frac{\text{no. relevant docs. presented to the user}}{\text{total no. docs. presented to the user}} \tag{6}$$

where Q is the query posted by the user. $R(Q)$ captures the fraction of relevant documents a search and retrieval algorithm is able to 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. Ideal performance is given by 100% recall and 100% precision.

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[5], despite the accuracy that it gives up by gossiping Bloom filters rather than the entire inverted index.

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

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. *Characteristics of the collections used to evaluate our search and retrieval engine.*

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.

4.1 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 [13] and includes full articles from Associated Press published in 1989.

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. We study a uniform distribution of documents because 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 [23]. We have also studied a community that may be representative of future communities based on PlanetP; 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.

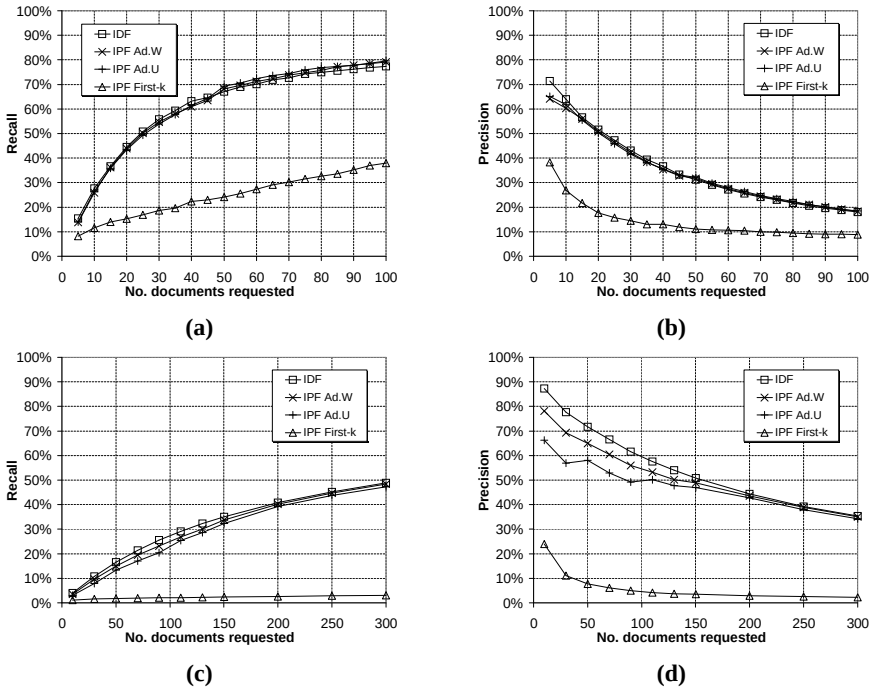

Fig. 1. *Average (a) recall and (b) precision for the MED collection distributed among 100 peers. Average (c) recall and (d) precision for the AP89 collection distributed among 400 peers. IDF is TFxIDF. IPF Ad.W is TFxIPF with the adaptive stopping heuristic on the Weibull distribution of documents. 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.*

Studying this community, we observed a data distribution that is very similar to that found by Saroiu et al., where 9% of the users are responsible for providing the majority of the files in the community. Using the collected data, we fitted a Weibull distribution with parameters ($\alpha = 0.7$, $\beta = 46$) and used it to drive the partitioning of a collection among a simulated community.

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 1 plots TFxIDF's and PlanetP's average recall and precision over all provided queries as functions of k for the MED and AP89 collections. 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 1 plotted the performance of PlanetP against k for a single community size: 100 peers for MED and 400 peers for AP89. In Figure 2a, 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 Figures 2c and 2d 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 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.

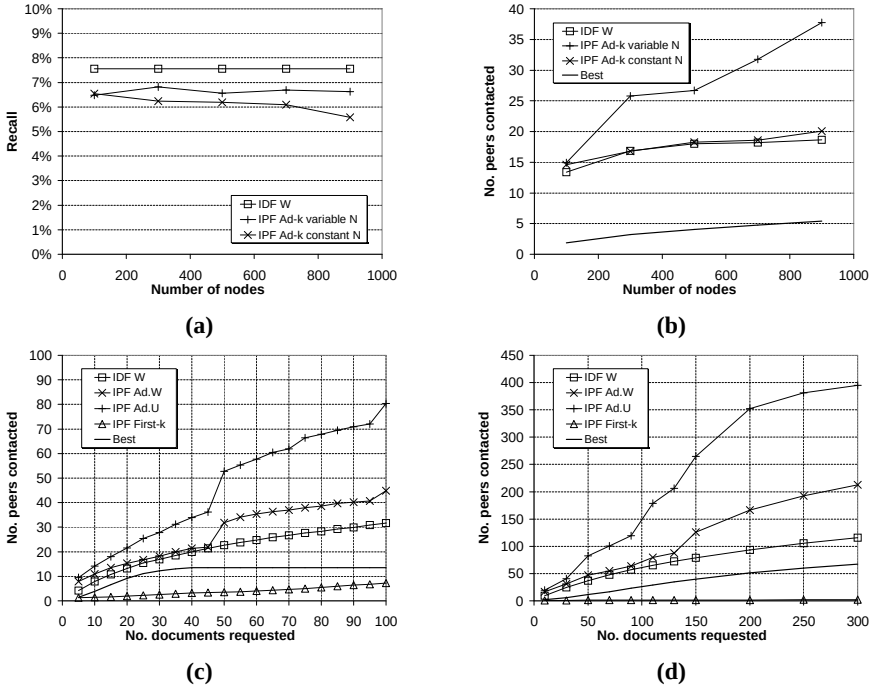

Fig. 2. *Average (a) recall and (b) number of peers contacted as a function of the community size (k=20). Number of peers contacted vs. k for (c) the MED collection distributed across 100 peers and (d) the AP89 collection distributed across 400 peers. IPF Ad-k variable N is TFxIPF with the adaptive stopping heuristic. IPF Ad-k constant N is TFxIPF with a stopping heuristic that is only a function of k and not of community size. 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 must be contacted to retrieve k relevant documents. All these plots use a Weibull distribution of documents except for IPF Ad.U, which uses a uniform distribution.*

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 TFxIDF. At corresponding $k's$, PlanetP's recall improves relative to TFxIDF: PlanetP outperforms TFxIDF slightly for MED and becomes essentially equal to TFxIDF. This implies that either equation 4 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.

Finally, we study the effect of making our adaptive stopping heuristic a function of community size; Figure 2b 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 figure 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.

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 relevant documents as TFxIDF. Comparing the sets of results returned, for the MED collection, by TFxIDF and TFxIPF at recall levels between 14% and 44%, we found intersections of 68% to 79%. We only studied the intersections for low recall values because at high recall, by definition, the intersection will approach 100%. Having, on average, an intersection close to 70%, indicates 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

While current P2P systems such as Napster [16], Gnutella [11], and KaZaA [14] have been tremendously successful for music and video sharing communities, their search engines have been frustratingly limited. 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.

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 [27], Pastry [21], Chord [24] and CAN [18] use different techniques

to spread (key, value) pairs across the community and to route queries from any member to where the data is stored. These systems differ from PlanetP in two key design decisions. First, in PlanetP, we explicitly decided to replicate the global directory everywhere using gossiping, which limits PlanetP's scalability. The advantage that we get, however, is that we do not have to worry about what happens to parts of the global hash table if members sign off abruptly from the community. Also, the entire community collaborate to spread information about what each peer has to share, instead of putting the publishing burden entirely on the sharing peer. Second, we have focused on content search and retrieval, attempting to provide a similar service to web search engines, which none of these systems have explored.

More related to PlanetP's information retrieval 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. 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. Both Gloss and Cori assume the existence of a server (or a hierarchy of servers) that will be available for users to decide which collections 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 on the presence of network and node failures.

6 Conclusions

P2P computing is a potentially powerful model for information sharing between *ad hoc* communities of users. 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. 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.

To explore content search and retrieval in P2P communities, we have approximated a state-of-the-art text-based document ranking algorithm, the vector-space model, instantiated with the TFxIDF ranking rule, in PlanetP. A naive implementation of TFxIDF would require each peer in a community to have access to the inverted index of the entire community. 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 summaries of individual indexes, (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. Our results provide evidence

that distributed content search and retrieval in P2P communities can perform as well as search and retrieval algorithms based on the use of centralized servers.

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

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. F. Douglis, A. Feldmann, B. Krishnamurthy, and J. C. Mogul. Rate of change and other metrics: a live study of the world wide web. In *USENIX Symposium on Internet Technologies and Systems*, 1997.
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. 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.
14. KaZaA. http://www.kazaa.com/.
15. 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.
16. Napster. http://www.napster.com.
17. A. Oram, editor. *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*. O'Reilly Press, 2001.
18. 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.
19. 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.
20. D. Roselli, J. Lorch, and T. Anderson. A comparison of file system workloads. In Proceedings of the 2000 USENIX Annual Technical Conference, June 2000.
21. 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.
22. 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.
23. 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.
24. 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.
25. I. Witten, A. Moffat, and T. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufmann, San Francisco, second edition, 1999.
26. 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.
27. 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.