pSearch: Information Retrieval in Structured Overlays

Chunqiang Tang[†]
Dept. of Computer Science
Univ. of Rochester
Rochester, NY 14627-0226
sarrmor@cs.rochester.edu

Zhichen Xu
HP Laboratories
1501 Page Mill Rd., MLS 1177
Palo Alto, CA 94304
zhichen@hpl.hp.com

Mallik Mahalingam
HP Laboratories
1501 Page Mill Rd., MLS 1177
Palo Alto, CA 94304
mmallik@hpl.hp.com

ABSTRACT

We describe an efficient peer-to-peer information retrieval system, pSearch, that supports state-of-the-art content- and semantic-based full-text searches. pSearch avoids the scalability problem of existing systems that employ centralized indexing, or index/query flooding. It also avoids the non-determinism that is exhibited by heuristic-based approaches. In pSearch, documents in the network are organized around their vector representations (based on modern document ranking algorithms) such that the search space for a given query is organized around related documents, achieving both efficiency and accuracy.

1. INTRODUCTION

The sheer quantity of Internet content is beyond the capability of any centralized search engine. A study [1] conducted by BrightPlanet Corporation in March 2000 estimated that the deep Web might contain 550 billion documents, far more than the 1.2 billion pages that Google had identified, not to mention the 600 million pages that Google was able to search at that time. Moreover, this data volume continues to grow at an astonishing rate, doubling each year. This calls for an infrastructure that is able to scale at a comparable rate.

Peer-to-peer (P2P) systems [8], on the other hand, are gaining popularity due to their scalability, fault-tolerance, and self-organizing nature, raising hope for building completely decentralized information retrieval (IR) systems.

Current P2P searching systems, however, are either unscalable or unable to provide deterministic guarantees. Usually they are based on one of the following techniques: centralized indexing, query flooding, index flooding, or heuristics. Centralized indexing systems such as Napster suffer from the single point of failure and performance bottleneck at the index server. Flooding-based techniques such as

---

[†]This work was done when the author worked at HP Labs in summer 2002. He is supported in part by NSF grants CCR-9988361, CCR-0219848, ECS-0225413, and EIA-0080124.

Gnutella send a query or index to every node in the system, consuming huge amount of network bandwidth and CPU cycles. Heuristics-based techniques try to improve performance by directing searches to only a fraction of the population, but they may fail to retrieve important documents.

Distributed hash table (DHT) systems such as CAN [9] do provide good scalability, but they only offer a simple interface for storing and retrieving (`key, value`) pairs, and hence are not suitable for full-text searches.

Besides the performance inefficiency, a common problem with most existing P2P systems is that they usually ignore the advanced ranking algorithms devised by the IR community through decades of refinement and evaluation, and thereby rely on naive keyword-based searches. Examples of successful IR algorithms include the vector space model (VSM) and latent semantic indexing (LSI) [2]. These algorithms represent documents and queries as vectors, and measure the similarity between a query and a document as the cosine of the angle between their vector representations.

Our goal is to build a scalable P2P IR system, *pSearch*, that has efficiency of DHT systems and accuracy of state-of-the-art IR algorithms. To achieve this goal, in pSearch, documents in the network are organized around their vector representations such that the search space for a given query is organized around related documents, achieving both efficiency and accuracy.

In this paper we describe two algorithms: pVSM that is based on the vector space model, and pLSI that uses latent semantic indexing. For pLSI, our simulation shows that it is able to achieve accuracy comparable to the centralized algorithm while visiting only 0.4-1% nodes in the system. For pVSM, the number of visited nodes is bounded by the number of terms in a query, which is usually small.

In pSearch, queries can be processed concurrently in different regions of the overlay, it hence has high throughput. It also has short response time, because nodes collectively search the partitioned document space to resolve a query. The performance of IR systems is usually bounded by disk I/O. In pSearch, indices are massively partitioned such that the indices held by a node may fit in its main memory.

Several features distinguish pSearch from other systems.

- It works in a completely decentralized manner, with neither single point of failure nor complex hierarchy.
- It supports content and semantic searches, as opposed to naive keyword matches.
- It is scalable, efficient, and effective. Index/query flooding are avoided. Its accuracy is comparable to that of the state-of-the-art centralized IR algorithms.

The remainder of the paper is organized as follows: Section 2 provides background information about DHT and IR. Section 3 describes pVSM and pLSI. We discuss pSearch applications in Section 4, present related work in Section 5, and conclude in Section 6.

2. BACKGROUND

pSearch is built on eCAN [12] (a hierarchical version of CAN) and uses extensions to VSM and LSI [2]. The Cartesian space abstraction of CAN makes it particularly attractive when used to store vector representations of documents generated by IR algorithms such as VSM and LSI. We describe these basic components below.

DHT systems and eCAN. Recent DHT systems, represented by CAN, Chord, and Pastry, offer an administration-free and fault-tolerant application-level overlay network. CAN stands for content-addressable network. It organizes the logical space as a d-dimensional Cartesian space (a d-torus) and partitions it into *zones*. One or more nodes serve(s) as owner(s) of a zone. An object `key` is a point in the space, and the object is stored at the node whose zone contains the point. Locating an object is reduced to the problem of routing to the node that hosts the object. Routing from a source node to a destination node is equivalent to routing from one zone to another in the Cartesian space. A node join corresponds to picking a random point in the Cartesian space, routing to the zone that contains the point, and splitting the zone with its current owner(s). eCAN improves CAN's logical routing cost to $O(\log(n))$, and takes only routes that closely approximate the underlying Internet topology.

Vector space model. VSM represents documents and queries as *term vectors*. Each element of the vector represents the importance of a word (*term*) in the document or query. The weight of an element is often computed using the *term frequency * inverse document frequency* (TF*IDF) scheme [2]. The intuition behind it is that two factors decide the importance of a term in a document: the frequency of the term in the document and the frequency of the term in other documents. If a term appears in a document with a high frequency, there is a good chance that the term could be used to differentiate the document from others. However, if the term also appears in a lot of other documents, e.g., `computer`, the importance of the term should be penalized. VSM usually normalizes vectors to unit Euclidean norm to compensate for differences in document length. During a retrieval operation, the query vector is compared to document vectors. Those closest to the query vector are considered to be similar and are returned. A common measure of similarity is the cosine of the angle between vectors.

Latent semantic indexing. Literal matching schemes such as VSM suffer from synonymy, polysemy, and noise in documents. LSI has been proposed to address these problems. It uses singular value decomposition (SVD) to transform and truncate a matrix of term vectors computed from VSM to discover the semantics of terms and documents. For instance, although `car`, `vehicle`, and `automobile` are different terms, LSI may be able to discover that they are related in semantics. Intuitively, LSI transforms a high-dimensional term vector into a medium-dimensional *semantic vector* by projecting the former into a medium-dimensional semantic subspace. The basis of the semantic subspace is computed
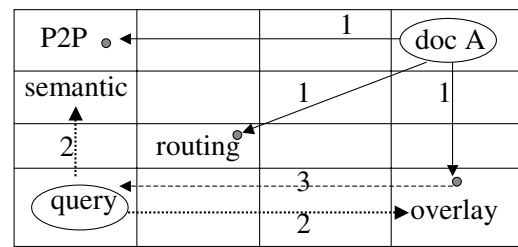

Figure 1: pVSM in a 2-dimensional CAN.

using SVD. Semantic vectors are normalized and their similarity is measured as in VSM.

3. PSEARCH ALGORITHMS

The fundamental idea in pSearch is to store information of documents in DHT-based overlay networks around their vector representations such that the search space for a given query is organized around related documents. We present two algorithms here: pVSM and pLSI. Each is described in terms of the basic ideas, challenges, and solutions.

3.1 Peer-to-Peer VSM (pVSM)

As is pointed out by Cuenca-Acuna and Nguyen [4], a straightforward implementation of VSM in DHT needs to store (`term`, `index`) pairs for each term appearing in each document, a formidable task simply due to the large number of terms in each document. Recent studies, however, show that the frequency of terms in a document usually follows a *Zipf* distribution, meaning that a small number of keywords can categorize a document's content.

Basic ideas. In pVSM, nodes are organized using CAN and each node is responsible for storing indices containing certain keywords. Given a document, a term vector is computed using VSM. The m most heavily-weighted elements (terms) t_i, $i = 1, ..., m$ are identified, and the corresponding ($h(t_i)$, `index`) pairs are stored in DHT. (m can vary on a per-document basis, Section 3.4.) Here h is a hash function mapping strings into points in CAN, and `index` is the actual document or a pointer to the actual document. During a retrieval operation, each term in the query is hashed into a point using h and the query is routed to nodes whose zones contain the points. Each of the nodes retrieves the best matching indices locally using VSM, and sends them back to the node initiating the query. The query initiator ranks them globally, discards those with low ranks, and presents the rest to the user.

Figure 1 illustrates this process. Each zone is owned by a node. The little dots represent indices. Given the document A, its important keywords—P2P, `overlay`, and `routing`—are identified using VSM and the index is published to corresponding nodes (*step 1*). Given a query of `"semantic overlay"`, it is forwarded to nodes responsible for keyword `semantic` and `overlay`, respectively (*step 2*). The two nodes then search and return matching indices using VSM (*step 3*).

Challenges and solutions. The first problem pVSM faces is synonymy. An index may be stored under one term but retrieved using its synonyms. This problem can be fixed using a thesaurus, by also routing queries to points corresponding to synonyms of the terms in the query.

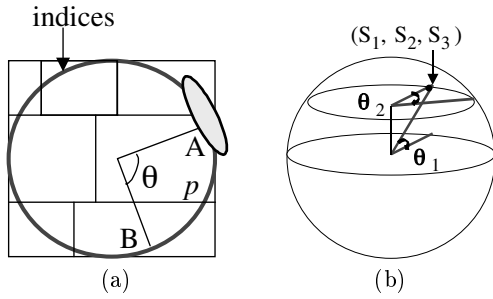Second, VSM relies on some global information, such as

Figure 2: Transforming the sphere space. (a) *In a 2-dimensional CAN, the naive pLSI only places indices on the circle. The similarity (cos θ) between the two indices A and B is proportional to their distance (p) on the circle: $\cos\theta = \cos p$. The gray region is the flooding area for searching indices close to A in semantics.* **(b)** *Using Equation 1 to transform a 3-dimentional semantic vector.*

the dictionary of the terms that TF*IDF counts, and the *inverse document frequency* (IDF). We call this global information *statistics*. It has been demonstrated that VSM does not need precise statistics to work well, i.e., a good approximation is sufficient.

In pVSM, the initial copy of the statistics are precomputed using samples that are representative of the potential document set. Over time, a combining tree that approximates the underlying network topology and includes randomly chosen nodes is used to sample documents and merge statistics. The size of the statistics grows slowly with the size of the document population. We expect the statistics to change slowly, at the rate of weeks or even months, because statistics are more stable than the documents themselves, especially for a large document population. The root of the combining tree periodically disseminates the statistics to other nodes. The root is a node occupying a well-known zone in CAN. If it fails, one of its neighbors will take over.

3.2 Peer-to-Peer LSI (pLSI)

In the original proposal of CAN, document IDs (keys) are randomly generated with no meanings other than their role in routing. If we use the semantic vectors of documents as keys to place indices in such a way that indices, that are semantically close to each other, are stored logically close to each other in CAN, then searching in the semantic space is reduced to routing in CAN. In the following sections, we first outline a naive pLSI algorithm to highlight the challenges that need to be overcome, and then describe the solutions.

3.2.1 A Naive pLSI algorithm

Let's use $\mathcal{L}$ and $\mathcal{K}$ to denote the LSI semantic space and CAN Cartesian space, respectively, with l and k as the dimensionality of the two spaces. Because l and k are freely tunable, we can map each document to a point in $\mathcal{K}$, by setting $l = k$ and treating its semantic vector as its coordinates in $\mathcal{K}$. Given a document, its semantic vector S is computed using LSI, and the (S, index) pair is stored in DHT using eCAN routing. During a retrieval operation, the semantic vector Q of the query is computed and the query is routed using Q as the DHT key. Upon arriving at the destination, it floods the query *only* to nodes within a radius r based on the similarity threshold or the number of wanted documents

specified by the user. All nodes that receive the query do a local search using LSI and merge the results back to the user as in pVSM. Because indices of documents similar to the query above the threshold can be stored only within this radius r and we do an exhaustive search within this area, pLSI achieves the same performance as LSI. Ideally, this radius r is small and the involved nodes are only a small fraction of the entire population.

3.2.2 Challenges for pLSI

There are four major problems that need to be solved.

- *Sphere distribution of semantic vectors.* Recall that semantic vectors are normalized and reside on the surface of the unit sphere $\mathcal{U}$ in $\mathcal{L}$, leading to an unbalanced load as depicted in Figure 2(a).

- *Uneven distribution of semantic vectors.* Even if the first problem is ignored and nodes are uniformly distributed on $\mathcal{U}$, it still suffers from hot spots because semantic vectors are not uniformly distributed on $\mathcal{U}$.

- *The curse of dimensionality.* Due to $\mathcal{L}$'s medium dimensionality (100-300), a straightforward nearest neighbor search in $\mathcal{L}$ will not be effective until a large fraction of the nodes are visited [11].

- *The global state problem.* Similar to pVSM, pLSI also needs some global information to work.

3.2.3 Transforming the Sphere Space

To solve the first problem, we transform a semantic vector $S = (s_1, s_2, ..., s_l)$, $||S||_2 = 1$ in $\mathcal{L}$ into a *parameter vector* $(\theta_1, \theta_2, ..., \theta_{l-1})$ in the $(l-1)$-dimensional polar subspace $\mathcal{P}$. Accordingly, we set $l - 1 = k$. Given a document or a query, we use the parameter vector, instead of the semantic vector, as the DHT key for eCAN routing. (To reiterate, we deal with three kinds of vectors: the *term vector* generated by VSM, the *semantic vector* generated by LSI, and the *parameter vector* generated by this transformation.)

This transformation does exist because points on $\mathcal{U}$ only have $l - 1$ degrees of freedom. Equation 1 achieves the exact goal. An example of this transformation is shown in Figure 2(b). Note that even after the transformation, parameter vectors are still not uniformly distributed in $\mathcal{P}$.

$$\theta_j = \arctan\left(\frac{s_{j+1}}{\sqrt{\sum_{i=1}^{j} s_i^2}}\right) \quad j = 1, ..., l-1 \qquad (1)$$

3.2.4 Balancing the Load

We use a modified node bootstrap process to solve the second problem. At node join, we randomly pick a document that the node is going to publish and use the parameter vector of the document as the random point towards which the join request is routed. This bootstrap process has three effects: (1) *Load balancing.* Each node stores roughly the same number of indices because the node distribution in $\mathcal{K}$ approximates the index distribution, given that a large number of nodes exist. [1] (2) *Index locality.* Assuming that a node's contents have some locality, the indices of its contents are likely to be published at itself or neighboring zones. (3) *Query locality.* Suppose that documents owned by a user

---

[1]Without the transformation in Equation 1, it can still achieve load balancing using a similar bootstrap process, but routing would be less efficient.
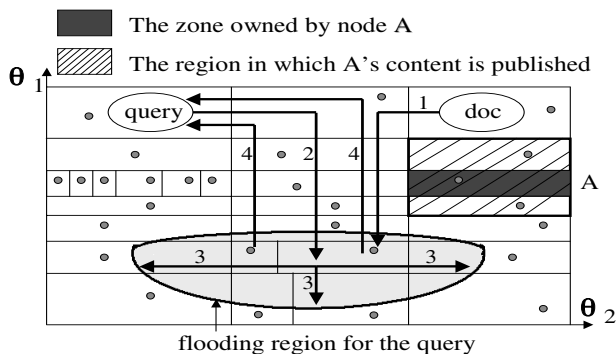
Figure 3: pLSI in a 2-dimensional CAN.

Figure 4: Multi-plane in a 2-dimensional CAN.

are good indications of the user's interests. Then queries submitted by the user would usually be searched in neighboring zones of where the query is submitted. Both the index and query locality lead to efficient eCAN routing.

Figure 3 depicts how pLSI works in a 2-dimentional CAN with the first two modifications. Given a document, its index is stored in DHT using its parameter vector as the key for eCAN routing (*step 1*). Given a query, it is first routed using its parameter vector as the DHT key (*step 2*) and then flooded to a small region computed from the given similarity threshold (*step 3*). Nodes in the flooding region search and return matching indices using LSI (*step 4*). Note that indices are not uniformly distributed but the number of indices per zone is roughly the same. Because of the transformation in Equation 1, the flooding radius r is not uniform in different directions. For node A, it is likely that its content is published in neighboring zones because of the *index locality* induced by the bootstrap process.

3.2.5   Dispelling the Curse of Dimensionality

Existing techniques [11] to relieve the curse of dimensionality usually do not work well at hundreds of dimensions, and require changing the DHT abstraction. In pLSI, domain-specific knowledge allows us to attack this problem at a lower cost. As a result of the SVD decomposition, the elements appearing earlier in a semantic vector are more important than those appearing later. This property also holds for parameter vectors. By aggressive cutoff, a parameter vector can be reduced to only a few dimensions to ease the nearest neighbor search, at the cost of precision.

We use a multi-plane scheme to reduce the dimensionality while keeping good precision. We partition lower elements of a parameter vector into multiple low-dimensional subvectors, with one subvector on each *plane*. The dimensionality of CAN is set to that of an individual plane. Each of the subvectors is used as the DHT key for routing. A document index is published to a node on each plane. A query is also routed and flooded on each plane. Given a query, each plane independently returns matching documents to the query initiator, based on subvectors on that plane. These returned documents form a *pre-selection set*. The query initiator then uses the *full* semantic vector to re-rank documents in this set. To increase the chance that relevant documents are included in the pre-selection set, we increase and vary the number of documents returned on each plane. For instance, when searching the top 15 documents for a query, the first three planes return 30, 20, and 10 documents, respectively,
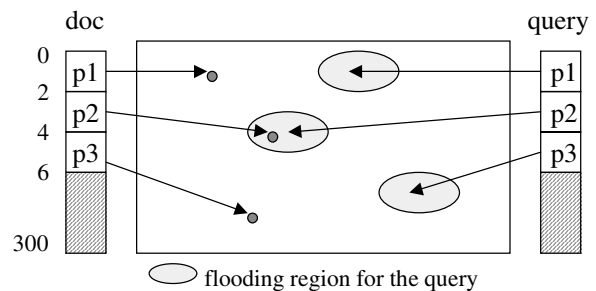
with a total of 60 documents in the pre-selection set.

A multi-plane example is shown in Figure 4. The first 6 elements of the vectors are partitioned into 2-dimensional subvectors on 3 planes. Both the document and query are routed on every plane. In this example, they match on the second plane. Note that only a single eCAN overlay is needed to support multiple planes.

3.2.6   Distributing the Global State

Similarly, pLSI also needs some global statistics: the dictionary, the IDF, and the basis of the semantic space. (Distributing the basis to each node allows the nodes to compute the projections from term vectors to semantic vectors independently.) The solution is also similar: precompute the statistics and update them using samples. When the basis shifts, the semantic vector of a document also changes. This may require redistribution of the index when the difference between two consecutive versions of a semantic vector is so significant that the old one and new one no longer reside in the same zone. Fortunately, since the statistics change slowly, this should happen fairly infrequently.

Computing SVD for a high-dimensional matrix is an intensive process. To avoid starting from scratch at every sample update, we incrementally update the matrix using *SVD-updating*. An alternative is to replace LSI with low-computation approximations such as *Concept Index*, and parallelize them, given abundant nodes in P2P networks.

3.3   pSearch Prototype and Initial Results

The SMART system, developed at Cornell, implements VSM. We extend it with a LSI module and link it with our eCAN simulator [12]. With the basic functionality of pVSM and pLSI in place, we experiment with two corpus widely used in IR research: the small MEDLINE corpus with 1033 documents and the large Text Retrieval Conference (TREC) corpus with 528,543 documents.

Our pVSM experiments running over the MEDLINE corpus demonstrate that storing a document index under the 30 most important terms in the document achieves a result as good as VSM. That is, if VSM returns a document for a query, with high probability, at least one of the terms in the query is among the top 30 terms in the document. The number of visited nodes is bounded by the number of terms in the query, which is usually small.

Since pLSI is more complex, we experiment it with the large TREC corpus, running a large number of configurations in search of the right system parameters. We find that using 4-6 planes with about 10-12 elements on each plane
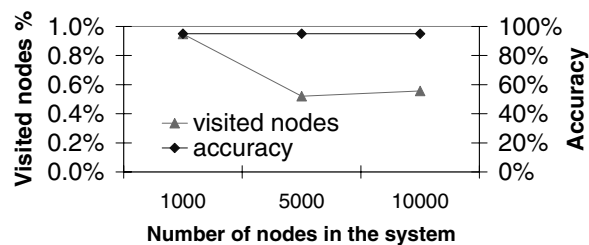
Figure 5: Performance of pLSI, using TREC corpus.

can be very effective. The result [2] is presented in Figure 5, where the X axis is the number of nodes in the system, the left Y axis is the percentage of visited nodes, and the right Y axis is the percentage of overlap between the documents returned by pLSI and LSI. We can see that pLSI is efficient as well as accurate. It only needs to visit about 0.4-1.0% nodes to achieve an accuracy of 95%. The accuracy is even better when more nodes are allowed to be visited. Due to the space limit, more results and our suggestions on configuring the system will be presented elsewhere.

Some differences between pVSM and pLSI are worth noticing. pVSM usually publishes more copies of an index than pLSI does, but it usually sends a query to less nodes during retrieval operations. The flexibility of pLSI allows it to be used in advanced searches such as video and audio retrieval (Section 4). In addition, pLSI can benefit from a modified bootstrap process for load balancing (Section 3.2.4).

3.4 Advanced Issues

We leave some advanced issues for future work.

Indexing at coarse granularity. For nodes that own a large collection of homogeneous documents, it is not necessary to maintain an index for each individual document. It has been shown that indexing at a per-database basis can work well [5]. In pSearch, we use *hierarchical k-means* to cluster documents at a node into collections until the variance inside a collection falls below a given threshold. Each collection is treated as a single document and its index is published in DHT. If a retrieval hits in a collection, the query is forwarded to the publisher for an in-depth search.

Relevance feedback. In addition to the conventional use of relevance feedback to build new queries, we also exploit user retrieval patterns to improve the index distribution. When answering a query, pSearch only returns a list of the best matching documents. It is the user who decides which document to download. Every node remembers the number of times that each document is downloaded and uses it as an estimation of document popularity. For a popular document, pVSM increases the number of terms under which the index is stored, to increase the chance of retrieving it with other queries. In pLSI, upon receiving a downloading request that originates from a successful query, the node records that the document index should move closer to the query vector. The move actually happens when this information is accumulated over a threshold.

Intelligent index replication. pLSI tries to balance

load by approximating the index distribution with the node distribution. However, nodes in the system usually have different processing power, storage capacity, and network connectivity. To take advantage of the heterogeneity in system, when the load at a node is low comparing with its capacity, it can selectively replicate indices stored on its direct and indirect neighbors, and process queries on their behalf. The criteria of this selection process may include the popularity of the documents, the load of those neighboring nodes, and user access patterns. The goals of the selective replication are to relieve hot spots and to maximize query performance.

Hierarchical refinement. Using a single semantic space to organize all documents in the universe will inevitably result in coarse-grained classification. There will be dense clusters in the semantic space, within which documents can not easily be distinguished from one other. This cannot be refined by simply increasing the dimensionality of the semantic space because the added dimensions may be helpful to some clusters, but introduce noise into others.

Our solution is to use the global statistics only as an initial step, and then compute localized semantic subspaces using localized information (e.g., only using documents whose semantic vectors fall in a particular region in this global semantic space) to refine the search. Intuitively, the global semantic space clusters documents into coarse areas such as computer science, geology and business, and local semantic subspaces further differentiate documents in each area. The challenge is perhaps to identify the dense clusters and coordinate the refinement process in a decentralized fashion.

Advanced searches. Our focus has been on extensions to the basic VSM and LSI algorithms. Many other searching techniques can also be adopted to our framework.

For instance, instead of querying the entire universe, one might only be interested in documents that reside in a particular domain (e.g., hp.com) or under a certain subdirectory, produced before or after a certain date, etc. Often, it is convenient if one can perform certain *join* functionality to query documents that related to an existing set of documents in a certain way. To provide these features, we suggest including other metadata such as domain, path, date, etc. in the index, to enable functionalities that are analogous to database selection and join.

Some IR systems exploit context information in documents for document ranking. For instance, given a query of `"computer network"`, documents with the two words close-by are ranked higher than documents with the two words far apart. We propose to use pVSM or pLSI to find a medium number of related documents, and then apply other ranking algorithms locally to re-rank them, if desirable.

Application-aware overlay. In Section 3.2.4, we balance the load by fitting the node distribution to the document distribution. This unfortunately may result in inefficient routing and high cost in overlay maintenance. Non-uniform node distribution can also be a result of topology-aware overlay construction and the discriminate use of nodes in terms of forwarding and storage capacity. An interesting problem is to allow the node distribution to fit the application's demand, while not sacrificing the performance of the overlay. Extending the ideas in eCAN [12], we propose to maintain only the basic routing states to make the overlay connected, and employ intelligent route caching based on application behavior to improve performance.

---

[2]The transformation in Equation 1 is not used in this experiment. It is tested seperately but not incorporated into the simulator yet.

4. OTHER APPLICATIONS

The pSearch technologies should be applicable to many other applications. We give only a few examples here.

Video/Audio. pLSI works by representing media contents as vectors and organizing contents around the vectors. This method can be applied to any media that can be abstracted as vectors and have its object similarity measured as some kind of distance in the vector space. A lot of pattern recognition problems fall into this category. For instance, people also employ SVD to extract algebraic features from images, and use various extractors to derive frequency, amplitude, and tempo feature vectors from music data.

Semantic-based Publish/Subscribe. Going one step further, pSearch provides a decentralized infrastructure for semantic-based Publish/Subscribe. The nodes are natural places for keeping document subscriptions and for document availability detection. The subscription can be described not only in topics and contents, but also in semantics, allowing users to subscribe to unstructured documents that they do not know how to describe precisely. pSearch users may simply describe their needs as *"notify me when documents similar to my collection show up"*.

Deep search in grid. Like the P2P model, the Grid also deals with resource sharing and cooperation among a large number of heterogeneous systems. To provide a uniform resource discovery and IR interface over existing heterogeneous services in Grid, we propose to use an overlay as the pSearch infrastructure to connect existing services. Nodes in the overlay maintain service indices and route queries, using the coarse-granularity indexing technique described in Section 3.4. For services that cannot provide the indices, pSearch either crawls their Web pages or uses *query-based sampling* to extract a good summary of the service contents.

Semantic-based resource discovery. pSearch, in essence, provides a decentralized resource discovery service, in which providers publish summaries of their resources and consumers use DHT routing to discover the resources. Both publishing and discovery can be expressed in either object IDs, contents, or semantics. A lot of applications can be implemented with this paradigm: P2P cooperative caching, interest-based online bidding, interest-based chat room, resource discovery in *ad hoc* networks, and so forth.

5. RELATED WORK

Both routing indices [3] and attenuated bloom filter [10] use heuristics to selectively forward queries to a subset of neighbors that are likely to contribute in resolving the query. A study by Lv et al. [7] shows that expanding-ring search and random walk are better than Gnutella's query flooding. All these systems try to improve performance by limiting searches to a fraction of the population. Due to the lack of control over the contents placement, they may fail to retrieve important documents.

Distributed IR systems such as GlOSS [5] usually employ a centralized or hierarchical index to direct queries. Cuenca-Acuna and Nguyen [4] follow the conventional database selection approach but use a bloom filter to summarize each node's contents and flood the network. JXTA search [6] is a query broker system built around centralized hubs. Currently, it does not address the problem of routing queries among hubs at a large scale.

6. CONCLUSION

We propose two algorithms, pVSM and pLSI, that combine the efficiency of DHT routing with the accuracy of state-of-the-art IR algorithms to offer advanced content- and semantic-based full-text searches. Our techniques avoid the scalability problem of systems that employ centralized indexing, or index/query flooding. It also avoids the non-determinism that is exhibited by heuristic-based approaches. To our knowledge, pSearch is the first system that allows decentralized, deterministic, and non-flooding P2P information retrieval based on contents and semantics.

To handle non-uniform distribution of documents in the semantic space, we propose a new node bootstrap process that achieves load balancing, index locality, and query locality. Furthermore, a multi-plane scheme is introduced to avoid inefficiency that can result from high dimensionality of the semantic space. We propose several uses of semantic-based indexing including semantic Publish/Subscribe, deep search in Grid, and so forth. We suggest several directions for future improvement, such as application-aware overlay.

Acknowledgements

References

[1] M. K. Bergman. The deep web: Surfacing hidden value. http://www.brightplanet.com/deepcontent.

[2] M. Berry, Z. Drmac, and E. Jessup. Matrices, vector spaces, and information retrieval. *SIAM Review*, 41(2):335–362, 1999.

[3] A. Crespo and H. García-Molina. Routing indices for peer-to-peer systems. In *ICDCS*, July 2002.

[4] F. M. Cuenca-Acuna and T. D. Nguyen. Text-based content search and retrieval in ad hoc p2p communities. In *Proceedings of the International Workshop on Peer-to-Peer Computing*, May 2002.

[5] L. Gravano, H. García-Molina, and A. Tomasic. GlOSS: text-source discovery over the Internet. *ACM Transactions on Database Systems*, 24(2), 1999.

[6] JXTA Search. http://search.jxta.org.

[7] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and replication in unstructured peer-to-peer networks. In *ICS'02*, New York, USA, June 2002.

[8] D. S. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, and Z. Xu. Peer-to-peer computing. Technical Report HPL-2002-57, HP Lab, 2002.

[9] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *ACM SIGCOMM'01*, August 2001.

[10] S. Rhea and J. Kubiatowicz. Probabilistic location and routing. In *INFOCOM'02*, 2002.

[11] R. Weber, H.-J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *VLDB*, 1998.

[12] Z. Xu and Z. Zhang. Building low-maintenance expressways for p2p systems. Technical Report HPL-2002-41, HP Laboratories Palo Alto, 2002.