A Scalable Semantic Indexing Framework for Peer-to-Peer Information Retrieval

Yan Chen
Computer Science Dept.
Northwestern University
Evanston, IL 60201

Zhichen Xu
Yahoo Inc.
701 First Avenue
Sunnyvale, CA 94089

ChengXiang Zhai
Dept. of Computer Science
University of Illinois at
Urbana-Champaign
Urbana, IL 61801

ABSTRACT

The exponential growth of data demands scalable and adaptable infrastructures for indexing and searching a huge amount of data sources with high accuracy and efficiency. Existing centralized search engines are not scalable and suffer from single-point-of-failures. The recent work on P2P index construction partitions the document vectors either randomly or statically, making it difficult to tradeoff between search efficiency and accuracy. In this *position paper*, we propose a peer-to-peer (P2P) IR framework (termed as *P2PIR*) that leverages a novel two-phase distributed semantic indexing on top of distributed hash tables (DHT). The distributed semantic clustering of P2PIR leads to good semantic locality on index placement so that the indices of similar documents are placed together or near to each other. The semantic locality enables smoother tradeoff between search accuracy and efficiency, as well as incremental adaptation to document and semantics changes. In addition, P2PIR allows for sophisticated retrieval techniques, *e.g.*, query refinement, feedback and personalized search for better usability.

A prototype of P2PIR is currently under development, which can be applied for general web retrieval and domain-specific applications such as a distributed electric medical records system.

1. INTRODUCTION

Recent years have seen explosive growth in the volume of information [36]. According to a recent study at University of California, Berkeley [29], the worldwide production of original content, stored digitally using standard compression methods, is at least 1 terabyte/year for books, 2 terabytes/year for newspapers, 1 terabyte/year for periodicals, and 19 terabytes/year for office documents. The exponential growth of data and information poses many challenges for information management, with a major challenge being the development of *scalable* infrastructures for indexing and searching a huge amount of data sources with high accuracy and efficiency.

The current web search engines are quite useful for helping people find information from the web. Unfortunately, they are all based on a *centralized* architecture, which is inherently unscalable. Indeed, all current search engines conceptually treat the whole web as a giant collection of documents and index them in a centralized manner. Thus in order to ensure freshness of the index, they all must periodically crawl the whole web to update the index. Such an updating would become more and more difficult and resource consuming as we have more and more information to manage, and eventually, the current centralized search model would break due to its unscalability. Another problem with the existing search engines is that they suffer from single-point-of-failure — when some critical centralized components fail, it would affect the availability of the whole system significantly, even cause a breakdown.

One way to overcome the deficiencies of the existing search engine architecture is to *distribute* search among the nodes in a network. Peer-to-peer (P2P) information retrieval is a new search paradigm that aims at exploiting such high-speed networks to perform search at every individual node in a network, leading to a potentially more scalable search infrastructure.

Most of the existing works in P2P information retrieval use term-index based approaches [27, 43] to support quick query execution, especially for short queries. However, an inverted-index alone can only support simple retrieval tasks and would be hard to support sophisticated retrieval algorithms which may require efficient access to all the terms in a document. Further, in a distributed environment, an inverted-index based approach would replicate document information in many places. Consequently, the results retrieved according to multiple keywords need to be intersected, consuming a large amount of network bandwidth [27, 43, 40].

In contrast to term indexing, document indexing can support many complex retrieval tasks because the information about the whole document is always available together. Recent work on pSearch leverages the document indexing and the emerging distributed hash tables (DHT) [37] for information retrieval on a P2P network [44]. But it has three shortcomings: 1) lack of semantic locality which leads to the degradation of search efficiency and accuracy; 2) unscalable index generation; and 3) static index generation which requires complete redo for adding documents with new concepts.

In this position paper, we propose a scalable semantic indexing framework for information retrieval (termed as "P2PIR") that can be regarded as an extension of pSearch to address its above-mentioned shortcomings.

The P2PIR framework extends pSearch in the following aspects:

- **Semantic locality:** A major extension to pSearch made in P2PIR is to use a novel two-phase distributed semantic indexing method to cluster documents that are similar in semantics into the same group and store them on nearby nodes on the grid system, which we refer to "semantic locality". This is in contrast to pSearch, which partitions document

vectors randomly. As a result, a query can be answered by searching on a small number of nodes, improving both query accuracy and efficiency.

- **Flexible tradeoff between search accuracy and efficiency:** The two-phase distributed indexing also enables us to tune the semantic clusters with various sizes, document density, and deployment density on the DHT, *i.e.*, a smooth tradeoff between search accuracy versus search efficiency and system overhead, which is necessary for managing huge amount of information.
- **Support of sophisticated retrieval methods:** Due to the possibility of local scoring of a working set of documents, P2PIR allows for sophisticated retrieval techniques, such as feedback and personalized search, to be computed efficiently, thus achieves high retrieval accuracy.
- **Adaptation to document dynamics:** The two-phase indexing enables P2PIR to incorporate new documents/concepts incrementally and efficiently without sacrificing search performance.

In the rest of the paper, we first describe the P2PIR architecture in Section 2. We then discuss each component of the architecture in Section 3 to Section 6. We survey the related work in Section 7, discuss our current status of prototype development in Section 8, and conclude in Section 9.

2. ARCHITECTURE

Figure 1 illustrates the architecture of P2PIR. P2PIR resides on a distributed hash table (DHT) based network. Applications such as medical record federation and document search can be built on top of it, as illustrated on the left part of the figure. The right part of the picture shows that P2PIR has two major parts: document indexing (shown on top) and querying (shown on bottom). (1) The document indexing part is responsible for converting all the source documents to some efficient data structures so that a query can be matched with all the documents quickly. A common practice in information retrieval is to extract features (e.g., words) that can represent the content of a document. P2PIR provides an open architecture, where different feature extraction algorithms, and similarity comparison algorithms can be plugged in. No matter what feature algorithm to use, the extracted features typically form a vector in a high dimensional space. A major challenge in indexing is how to store these vectors so that later they can be looked up quickly when we execute a query. (2) The querying part is to match a query against all the feature vectors of documents and score the documents. Again, P2PIR provides an open slot for plugging in any query refinement components that can take advantage of user feedback and improve retrieval accuracy. We now discuss these two major parts in detail.

Next, we will first introduce DHT, and then sketch the two-stage document indexing and query processing.

2.1 Distributed Hash Tables (DHT)

The term "DHT" is a catch-all for a set of schemes sharing certain design goals ([37, 42, 47, 15], *etc.*). As the name implies, a DHT provides a hash table abstraction over multiple distributed nodes, such as all the distributed nodes that form the P2PIR system. Each node can store indices (pointing to the documents), and each index is identified by a unique key, called *index locator*. The DHT systems provide two basic interfaces:

Insertion (`put`): The interface for insertion, `put(key,object)`, causes the DHT to route the given `object` to the node with a node identifier closest to the `key`.

Retrieval (`get`): The interface for retrieval, `object=get(key)`, causes the DHT to obtain the `object` from the node with a node identifier closest to the `key`.

The `objects` and `keys` in the P2PIR system are further discussed in Section 2.2.

DHTs provide strong theoretical bounds on both the number of hops required to route a key request to the correct destination, and the number of maintenance messages required to manage the arrival or departure of a node from the network. Such properties help provide scalability for routing, indexing and location of the P2PIR system. In addition, DHT provides some very nice properties for distributed systems, such as fault-tolerance, robustness [37, 42, 47, 15], and DoS attack resilience [8]. By contrast, early work on P2P routing used "unstructured", heuristic schemes like those of Gnutella and KaZaA, which provides no such guarantees: they can have high routing costs, or even fail to locate a key that is indeed available somewhere in the network.

In addition to having attractive formal properties, DHTs are becoming increasingly practical for serious use. They have received intense engineering scrutiny recently, with significant effort expended to make the theoretical designs practical and robust.

2.2 Two-stage Document Indexing

P2PIR indexes documents in the following two steps. In the first step, the entire document corpus is clustered into clusters with appropriate sizes and density. (This can be accomplished using either top-down or bottom-up methods.) The clusters can be disjoint or overlapping. In Section 4.1, we describe how clustering can be accomplished in a distributed and incremental fashion using concept indexing [23]. Each cluster correspond to a *concept*. There is a unique *concept vector* produced for each document.

In the second step, all the clusters are mapped on to the same distributed hash table (DHT) based network, in a way that semantically similar documents are placed close to each other on the network. As discussed before, we will use DHT (`put`) and (`get`) operations for index insertion and retrieval. In the P2PIR system, the `object` is the index (location) of each document, and the `key` should uniquely identify the document, *i.e.*, ideally the concept vector of the document. But the dimension of concept vector usually is much larger than that of the DHT, *i.e.*, we cannot fit a concept vector into a key of the DHT.

To solve the dimension mismatch problem, instead of randomly dividing these concepts into chunks (like pSearch [44]), we group them based on the aforementioned clusters so that related concepts will have a large probability to fall in the same group. For example, the concepts "manufacturer", "insurance", "car reviews" may fall in the same group as "automobile"; but they will fall into different group from that of the "computer network" which may contain concepts such as "security", "protocol", and "application". We limit the size of each group to be no larger than the size of the DHT dimensions.

Then we project the concept vector onto the subspace defined by each concept group. We call the resulting vector *index locators* which is used as DHT keys when we assign the concepts of that group to each dimension of the DHT. Take the index locator of automobile concept group for example, "automobile" may be assigned to dimension one, and "manufacturer" and "insurance" may be assigned to dimension two and three, respectively. Documents that are similar in contents are placed close to each other on the DHT because they have similar index locators.

One document can have multiple index locators because it may involve multiple groups of concepts, and we will insert an index for each of these groups. Figure 2 illustrates the basic idea of mapping

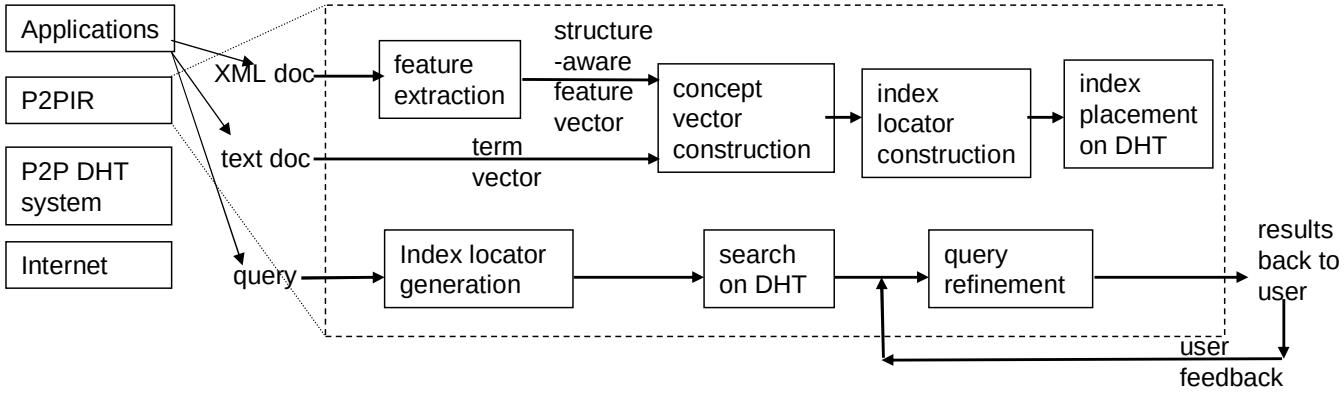

Figure 1: Layered model and architecture of the P2PIR system.

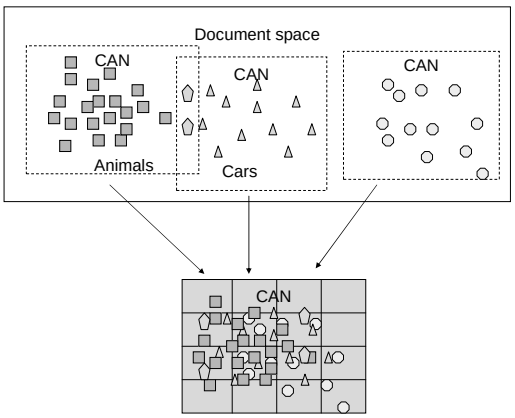

Figure 2: Document clustering and placement.

Figure 3: Query processing.

multiple clusters onto the same DHT overlay and document placement based on content similarity. In the figure, document clusters "animals" and "cars" are mapped onto the same DHT. There are some concepts overlap in the "Animals" and "cars" clusters, such as "Jaguar".

2.3 Querying

When querying, a user can contact any node on the DHT overlay. This node is responsible for projecting the query vector onto the concepts of each cluster, to produce a number of index locators. Because a query vector typically only involves limited number of clusters, we can expect to only have a few index locators, and in many case just one cluster. Each of the (index locator, query vector) pair is then routed in the DHT (in parallel) using the index locator as DHT key. Upon reaching the destination, the query vector and original document vectors are used to perform relevance ranking, and this local retrieval process can expand to neighboring DHT nodes until enough relevant results have been identified. Figure 3 illustrated the query process.

Note that the projection of the original query vector into index locators identifies the clusters that potentially have the matching documents. Then the routing and local search process identify the matching results of the query in each of the clusters.

For example, in Figure 3, the query terms "Jaguar+speed" are mapped to two concept groups (animal name, velocity) and (car name, velocity). Projection onto these concept groups produces two index locators. This projection phase, in effect, identifies the two clusters "cars" and "animals" that potentially have matching documents. The index locators are used for searching inside the two corresponding document clusters in parallel, respectively.

In summary, the novel two-step distributed indexing framework provides P2PIR with the following properties.

1. **Index placement with good semantic locality.** Such locality can significantly improve the retrieval accuracy and efficiency. For example, it allows us to match two related but distinct terms.

2. **Tunable framework and flexibility** The document corpus can be divided into clusters which different sizes and density. Furthermore, the space on the DHT to store a cluster can be shrunk or enlarged depending on the density of the cluster. These features provide P2PIR a smoothly tradeoff between the retrieval accuracy and efficiency (including storage overhead).

3. **Incremental adaptation to document/concept dynamics** The two-step procedure also allows us to change the dimensionality of the concept vectors without modifying that of the index locators. The latter is determined by the DHT, which tends to be more static. This property is important for efficiently adding and removing documents incrementally.

In the most degenerated case, we can use a one-dimensional DHT, and map each term (or concept) to a different point on the DHT, which is exactly how a typical search engine does the inverted index. Alternatively, we can select similarity-preserving hash functions (SPH), such as space-filling curves [1], the random permutation function by Broder et al. [2], *etc.*, to eliminate the dependence the geometry of the underlying logical abstraction of the DHT.

Note that the P2PIR architecture only makes the following two assumptions about the underlying retrieval models, and can potentially support many different retrieval models. (1) Documents can

be represented as vectors. (2) Euclidean distances are reasonably accurate in capturing document topic similarity. Both are quite reasonable. Most existing retrieval formulas are based on a vector representation of documents. This is not only true for the traditional vector space model, but also true for probabilistic models where a document is often represented as a word distribution, which can also be stored as a weight vector. While many known effective retrieval formulas are based on non-Euclidean distances, with appropriate weighting and normalization, Euclidean distances can be reasonably accurate. Since the Euclidean distances are only used to prune non-promising documents, its accuracy only indirectly affect the final retrieval results.

Next, we will describe each component of the P2PIR system in more details.

3. FEATURE EXTRACTION

The feature extraction module is responsible for generating a set of features and their values from a document. These feature values form a feature vector that serves as a representation of the document for the purpose of matching a query with a document. In the simplest case, the features can be just words in the document and feature values can be the frequency counts of words in the document. Such a simple representation (often called a "bag-of-word" representation) is often the basis of many current retrieval algorithms. More advanced approaches can enrich word-based representation by extracting multi-word phrases.

The features extracted normally serve *directly* as the indexing units for matching features extracted from a query with those extracted from documents. Since the number of features (e.g., words) is usually extremely large, direct indexing of such features makes it inefficient to match features in a distributed environment. One important advantage of the proposed P2PIR framework is precisely to further impose structures on the feature space so that we can manage the indices in a high dimensional space efficiently without sacrificing much accuracy. Specifically, P2PIR does not impose any constraints on the choice of features and the assignment of their values. Indeed, it is sufficiently general to support any kind of feature vector representation of documents, thus can support any retrieval algorithm as long as the algorithm is based on some kind of feature vector representation.

For example, it is possible to model an XML document more accurately by distinguishing features extracted from different parts and assign weights according to heuristics. The following are some useful heuristics: (i) In documents such as HTML or XML, terms that appear in titles and anchors usually carry more weights than those in other positions. (ii) In HTML documents, anchor texts usually describe the referred documents instead of the current document. (iii) The terms and phrases in section headers are usually more important than terms and phrases in lower-level headings and document body. (iv) Terms, phrases, and sentences that appear in the abstract, introduction, and conclusion, and the beginning/end of paragraphs usually carry more weights. Those appearing in related work and background sections of a paper could refer to the background that may not necessarily be the main focus of the work. (v) In HTML documents with frames, terms that appear in the different frames usually carry different weights. All these heuristics can be easily supported in the proposed P2PIR framework. Since heterogeneous documents can often be represented by XML documents, P2PIR has a great potential to support algorithms that deal with heterogeneous documents.

Web search algorithms may also exploit hyperlinks between documents to improve retrieval accuracy. A well-known algorithm is the PageRank algorithm which is used in the Google search engine [34]. Such algorithms can typically be implemented *separately* from the content-based scoring part, which is the focus of P2PIR. Thus we can easily add such scoring methods on top of P2PIR. Although the scalability of such scoring algorithms may itself be a very interesting research topic, it is outside the scope of this paper.

4. DISTRIBUTED SEMANTIC INDEXING

With the feature vector extracted, in this section, we discuss how to construct concept vectors and index-locators, and how to place indices on the DHT.

4.1 Concept-vector construction

While indexing documents directly with keywords has been a standard practice in information retrieval, it clearly has several limits: (1) Words with a similar meaning but different forms (e.g., *automobile* and *car*) will not match each other, which often decreases the recall. (2) A word with multiple meanings (e.g., *jaguar*) in the query may retrieve noisy documents, which decreases the precision. (3) Indexing with keywords involves an extremely high dimensional space, which is not desirable from the perspective of scaling up retrieval — in terms of both space and time. Thus one idea in P2PIR is to construct a concept space with a much lower dimension than the raw vocabulary, and map documents into this low-dimension space.

Latent Semantic Indexing (LSI) [14] applies singular value decomposition (SVD) of the whole $\langle$term document$\rangle$ matrix to extract the main components. Unfortunately, SVD is computationally expensive, and is clearly not scalable given that we have to work with an extremely large number of documents. Furthermore, when adding/removing documents, it becomes hard to incrementally update the SVD results. Recent probabilistic approaches improve upon the efficiency of LSI (e.g., [21]), but they are still too computationally expensive to scale.

We plan to use more efficient concept indexing approach for constructing the concept space [23]. The basic idea is to group documents into k clusters, and the centroid of each cluster corresponds to a concept. Then we use the concepts to index the documents as follows. Given a document d, the similarity between its feature vector and a concept c (e.g., cosine value between them) defines the weight of d on concept c. Then the concept vector of d is composed of its weights on all the concepts.

The flowchart for concept vector construction is shown in Figure 4. For document clustering, we propose to use a modified k-means clustering method. The k-means clustering method iteratively improves a hypothesized partition of documents, thus can stop at any time and still generate complete clustering results, which is a desirable property for clustering a large number of documents. Moreover, each iteration involves only $O(kN)$ document comparisons, where k is the number of clusters and $N >> k$ is the number of documents, as opposed to $O(N^2)$ document comparisons in a regular agglomerative hierarchical clustering method.

4.2 Index locator construction

Although the dimensionality of concept vectors can be significantly lower than that of their feature/term vectors, it could still be much higher than that of the index locators. To reduce the dimensionality, we divide a concept vector into multiple chunks, and each chunk has the size no more than the dimensionality of the underlying DHT. Thus one concept vector becomes multiple index locators.

If we randomly divide concept vector into index locators, the related concepts are probably in different chunks. Consequently, a

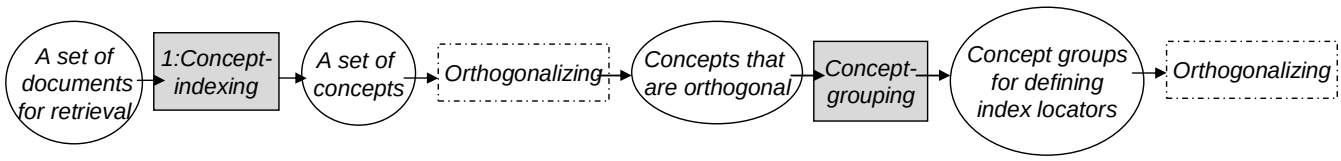

Figure 4: Extraction and clustering of concepts to define a set of index locators.

document will need more indices, and indices of related documents tend to be more scattered, causing inefficiency and inaccuracy of querying.

We propose to cluster the concepts into semantically coherent groups. Each group consists of concepts that are closely related (*e.g.*, "computer sciences" and "computer engineering"). Thus each concept group defines an index locator, and has its size no more than the dimensionality of the DHT. The grouping can be executed at the tree root node of the hierarchical clustering for concept vectors. Then the root node disseminates the concepts (represented by the feature vector of each cluster centroid) and the index-vector-generation rules (represented by concept clusters) to all other peer nodes with application-level multicast.

4.3 Optional orthogonalizing phases

Each of the two steps described above can be followed by an optional concept orthogonalizing phase. This phase can further reduce the dimensionality of the concept space, and improve the quality of the index locators.

The basic idea of this optional phase is to find a set of orthogonal axes to be used to project each vector into a lower-dimensional space. We would like to compute the axes in a distributed fashion within each peer, therefore, we plan to apply the FastMap algorithm [18] for its high efficiency and effectiveness. To find d mutually-orthogonal axes, where d is the reduced dimensionality, FastMap iteratively chooses two objects (called "pivot objects") to form an axis. For each axis, FastMap projects all objects onto this axis by computing new their coordinates. Conceptually, it also computes the remaining distance matrix after the projections. (This computation is needed only for those objects that are used in the subsequent steps.) The algorithm iterates d steps to find the axes.

4.4 Index placement on the DHT

To store the document indices in the system, each node computes the concept vector and the corresponding index locators for each assigned document. For each index locator of a document D_i, if its norm (i.e., length of the vector) is over certain threshold, we put an index of D_i along with its feature vector on the peer node whose DHT address vector matches best with the index locator.

Such a scheme is particularly appealing for index locators generated by the concept-grouping approach for the following two reasons. (1) The number of document pointers is reduced. Since we group related concepts into the same index locator, each document will have substantial weights in only a small number of index locators. And we only need to deploy indices with these index locators. (2) Semantic locality is achieved. Similar/related documents have similar index locators, and thus are placed on the same or nearby DHT nodes.

For instance, a document d on hybrid car has substantial weights only on two concepts: car/automobile and energy/power. These two concepts are related and may be grouped into the same index locator. Then we only need to place one index for d on a DHT node with address decided by the index locator.

5. QUERY PROCESSING & REFINEMENT

5.1 Query processing and semantic routing

P2PIR can accept keywords or a sample document (in the case of query-by-example) as a query input. In both cases, we will first extract features from a keyword query or an example document in a way similar to how we extract features from the documents in our collections. We then route the query feature vector to clusters that most likely contain documents similar to the query and search within such most promising clusters. We now discuss query routing and searching within clusters in detail.

5.1.1 Query routing

After we obtain a query feature vector, we will follow the following procedure to route the query to the most promising clusters of documents: (1) Project the query feature vector to the concept space to obtain (possibly multiple) query index locators via techniques discussed in the previous section. This step mainly involves computing the distances between the query and each cluster centroid. (3) For each index locator v_i of the query, the system routes the query in the overlay network using v_i as the DHT key. Each destination node answers the query by looking up its local documents and propagating the query to neighboring nodes. The projection process is based on the whole query, thus it can correlate the keywords in the same query to reduce semantic ambiguity.

Compared with pSearch [44], our query processing procedure provides much better query efficiency and accuracy for the following reasons. First, in our system, we partition the concepts (produced via concept indexing and the optional orthogonalization and reduction) into groups. Related concepts in a query can be expected to belong to a single group, thus the query needs only a single index locator. This can significantly reduce the number of query messages. Second, we place the indices according to semantic distances between documents. During a query-routing process, at each hop along the path towards the destination, the documents which have deployed their indices become more and more semantically relevant to the query. We refer to this property as *semantic routing*, which can find areas in the overlay with indices for related and similar documents quickly. Thus the desired documents can be efficiently retrieved by simply checking each destination node and a few of its direct neighbors. Third, with such semantic locality, query refinement (Section 5.2) may be achieved within a few hops of each destination node of the query in DHT.

Note that semantic routing requires two conditions: 1) placement of document indices with good semantic locality, e.g., through grouping related concepts, and 2) wildcard search for which any term in the query vector can be a range or a regular expression. Interestingly, we found that wildcard search can only be achieved efficiently with a hypercube geometry of a DHT (e.g., CAN [37]) without flooding. A hypercube geometry treats all concepts equally, whereas other structures (e.g., trees) require matching the concepts in a predetermined order. An important research direction is to further investigate how to enable wildcard search in other types of DHT.

In the remaining of the paper, we use CAN as the underlying DHT for building P2PIR. CAN [37] partitions a d-dimensional Cartesian space into *zones* and assigns each zone to a node. Two nodes are routing neighbors in the overlay if their zones overlap in all but one dimension along which they abut each other. An object key is a point in the Cartesian space and the object is stored at the node whose zone contains the point. Locating an object is reduced to routing to the node that hosts the object. Routing translates to traversing from one zone to another in the Cartesian space.

5.1.2 Searching within clusters

After we route the query vector to the most promising clusters, we would search within these clusters of documents to generate a ranked list of documents in the order of relevance to the query topic. One important advantage of our P2PIR is that we *decouple* searching within clusters from routing queries to the right clusters, which has at least two benefits: (1) Computation is highly parallelized – query routing for one query can be in parallel with searching within clusters for another query, and searching in different clusters can be performed in parallel in all cases. (2) Retrieval methods can be different for searching within clusters and routing queries — since searching within clusters can be done in parallel and we can control the size of clusters, we can afford using more complex retrieval methods for searching within clusters, which may directly affect the retrieval results. Thus in principle, we can support any retrieval methods, including link-based scoring methods like PageRank [34]. Since final scoring happens locally, we may also consider precomputing the link-based scores and partitioning them in a way similar to how we partition our indices.

5.2 Query refinement

Query refinement is an important technique for improving retrieval accuracy. We now discuss how our architecture can naturally support several query refinement strategies.

One standard technique for query refinement is through feedback [41]. In *relevance feedback*, the user is asked to make judgments on some of the retrieval results, which can then be exploited to update the query feature vector. This technique has been shown to be very effective for improving retrieval performance [38]. Intuitively, this is easy to understand. Consider a short ambiguous query such as "jaguar", which likely returns both documents about cars and documents about the animal. But if a user can tell us which documents he/she likes, we would be able extract some terms from the positive examples that can distinguish positive examples from negative ones and add them to our query so that our query now is more tuned to retrieve documents with either the car or the animal.

Since not all users are willing to judge documents, we may also simply assume the few top-ranked documents in our initial retrieval results to be relevant, and assume a random sample of documents from the whole collection to be non-relevant. This is called *blind (pseudo) feedback*, and has also been shown to be effective in improving retrieval accuracy in general [17, 3, 45, 46].

Regardless which form of the feedback, the system would utilize the feedback information to modify the query vector and issue a refined query. This process involves frequent access to document vectors and relatively sophisticated computation. One reason why such feedback techniques have not been supported in existing retrieval systems is because such computation cannot be done efficiently. P2PIR can support such index access and computation much more efficiently as the document index is stored by semantic locality and feedback documents are often semantically related.

To further improve retrieval accuracy, we may also refine a query using any search context or additional user information. One major limitation of the existing search engines is that the retrieval decision is generally based solely on the query and document collection; information about the actual user and the search context is largely ignored. As a result, although different users may use exactly the same query (e.g., "java") to search for different information (e.g., for the programming language or for coffee), an existing IR system would return the same results for these different users. It is thus highly desirable for a retrieval system to incorporate both *user information* and *search context* into the retrieval decision process.

For instance, though a text query is typically short, it is rarely the case that a user would find all the needed information with just one query; it is often necessary to formulate the query in order obtain better retrieval results. In general, we may have a sequence of previous queries available to us, which we can exploit. Also, when a user receives retrieval results, a user may view some documents but not some others, which can also provide extra information about what the user likes. Both the previous queries and the documents viewed and not viewed can be treated as the search context of the current query, and such search context can be exploited to improve the retrieval performance of the current query.

Exploiting search context and user information requires a great deal of computation. With the P2PIR framework, such computation can be performed on different machines in parallel, especially if we put more computations on a user's machine. For example, we can develop a sophisticated user-agent that keeps track of a user's search history and perform query refinement through these agents. The same agent can also re-rank or re-organize the search results to better satisfy a user's information need.

6. ADAPTATION TO DYNAMICS

All the entities in P2PIR can change continuously, including documents (their semantics), nodes, and networks. We focus on techniques to handle document dynamics. Our goal is to make the system self-organizing and evolving, and require little human intervention for dynamic document insertions, deletions, and updates.

As the documents keep changing, more concepts become available, and existing concepts disappear. Thus the dimensionality of the concept vectors can change. As this happens, an important goal is to keep the existing index locator as intact as possible.

The basic idea is to introduce new concept groups when necessary, and periodically overhaul the system to rebuild index locators.

Specifically, when a set of new documents emerge, we check (1) whether they contain new frequently-used terms or new heavyweight terms; and (2) whether their concept vectors belong to any existing cluster in the existing semantic space. If they have no new important terms and belong to some existing clusters, we just index them using existing concept dimensions. Otherwise, we will generate one or more new concepts for the documents, and expand the concept semantic space.

To add and index a new concept c, if c belongs to an existing concept cluster whose size is less than that of the underlying DHT, we can add c to that cluster by using the next available entry of the index locator. Otherwise, we generate a new concept group and a new set of index locators to represent c, and wait for infrequent redo to consolidate the two groups. Note that for both cases, we do not need to change the indices or the index locations of existing documents because existing documents have value zero on this new concept dimension by default. Then we generate the index locators for the new documents, and deploy their indices on DHT. Finally, we multicast the addition of the new concept c, and the addition of new concept group (i.e., the new set of index locators) when applicable, to all P2P nodes, so that they can route queries about

c. To avoid collisions from multiple P2PIR nodes, we have a small number of nodes to control new concept generation.

For instance, when new documents on "Bin Larden" appear, we detect it as a new concept relating to the concept group "terrorism". If the dimensionality of DHT is 20, and the size of "terrorism" concept group is 17, we can just add "Bin Larden" to that group as dimension 18 of the index locator. The corresponding index locators of existing documents have weight zero as default on dimension 18, and thus remain the same. If the terrorism concept group is already full, we generate a new concept group for "Bin Larden"(i.e., a new set of index locators).

Note that the quality of the index locators could gradually lose the semantic locality. Since the number of new concepts generated within a short period of time could be much smaller than that of existing concepts, the locality degradation is slow. Thus we envision another relatively infrequent process that will redo the indexing to improve the overall quality of the indexing, which could be triggered by detecting the distortions.

7. RELATED WORK

The related work is too numerous to enumerate, and we only list those that are most related to our work.

Feature Extraction: Feature extraction has been studied intensively in information retrieval, *e.g.*, [25, 31]. Our proposal is flexible to work with most existing feature extraction methods which represent a document as a weight vector. These also include dimensionality reduction methods such as LSI [14] and structure synopses [30], and XML document indexing methods, *e.g.*, [28, 11].

Distributed Semantic Indexing: There has been a huge body of work on building indexing structures to conduct similarly search in a high-dimensional space [26, 22, 19, 39]. Our work differs from these work in that documents and their index pointers are *distributed*, and the search process is conducted in different peers, and ideas from existing work can be leveraged to design an efficient distributed search strategy; indeed some of our ideas are related to MDS [24], Karhunen-Loeve ("K-L") transform [16], Local Dimensionality Reduction [5], and FastMap [18]. Work on distributed information retrieval (e.g., [20, 4]) is also related, but a centralized retrieval interface is assumed, which is less robust and less efficient than the proposed P2PIR architecture.

The proposed dynamic adaptation of an index has not been studied in the previous work.

Query Processing and Refinement: P2P Query processing has been extensively studied [13, 12, 9, 7, 10, 27]. A main deficiency of existing work is that most systems use simple keyword matching, and cannot support the advanced relevance ranking algorithms. Refinement and feedback of query have been active research in information retrieval [41]) and have received much attention in the database community (*e.g.*, [33, 6]). P2PIR distributes document indices to different nodes according to their semantic locality, and it can effectively distribute the extensive computation that are associated with these query refinement and feedback tasks.

8. STATUS

We are currently developing a prototype of the P2PIR system and and evaluating it with the Text Retrieval Conference (TREC-7 and TREC-8) corpus, a large test set widely used in IR research. It includes 528,543 documents from news, magazines, *etc.*, with a total size of about 2GB. Furthermore, we plan to deploy the P2PIR system on the PlanetLab testbed [35], and evaluating with the following two specific application domains.

Web retrieval: We are using variable-size subsets of Web data to quantitatively evaluate the scalability, search efficiency, search accuracy, and usability of P2PIR. We will further demonstrate how we can make smooth tradeoff between search accuracy and overhead in P2PIR, and how to adapt to document/concept dynamics. The effectiveness of P2PIR in handling semi-structured and structured documents will be evaluated in the medical record application below.

Distributed Electronic Medical Record Systems: Collaborating with the Radiology Department at Northwestern University, we are applying P2PIR for creating, indexing and searching of distributed electronic medical records. One major problem that exists by virtue of the complexity of the health-care environment is that there is no single information system that can or will ever meet all of the needs of all of the participants in the many health-care processes. P2PIR not only facilitates the storage of the pointers in the distributed grid of registry nodes, but would also facilitates retrieval of all the object pointers of heterogenous document formats (*e.g.*, DICOM [32]).

9. SUMMARY

In this position paper, we extend a state-of-the art P2P indexing method (i.e., pSearch) and propose P2PIR, a distributed tuneable IR systems that leverages the emerging DHT technologies. A main idea of extension is to use a novel two-phase indexing method, in which semantic clustering is used to achieve good semantic locality, search efficiency, search accuracy, tunability and agility to document/concept dynamics. The P2PIR can potentially support query refinement, feedback and personalized search, which can all help to improve the usability. We have sketched the basic ideas of P2PIR; many research questions on how to implement these ideas remain open for further study. We are now in the process of developing a prototype P2P retrieval system based on the proposed architecture.

As more and more information becomes available online, scalability becomes critical. P2PIR can be applied to manage a huge amount of information with flexible tradeoff between search accuracy and efficiency. Naturally, as a general retrieval architecture, P2PIR has many different applications, such as bioinformatics, web search, and other information and data management problems.

10. REFERENCES

[1] A. Andrzejak and Z. Xu. Scalable, efficient range queries for grid information services. In *Proc. of 2nd IEEE International Conference on Peer-to-Peer Computing*, September 2002.

[2] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher. Min-wise independent permutations. In *Proc. of 13th annual ACM symposium on Theory of Computings*, 1998.

[3] C. Buckley. Automatic query expansion using SMART: Trec-3. In D. Harman, editor, *Overview of the Third Text Retrieval Conference (TREC-3)*, pages 69–80, 1995. NIST Special Publication 500-225.

[4] J. Callan. Distributed information retrieval. In W. B. Croft, editor, *Advances in Information Retrieval*, pages 127–150. Kluwer Academic Publishers.

[5] K. Chakrabarti and S. Mehrotra. Local dimensionality reduction: A new approach to indexing high dimensional spaces. In *Proc. of VLDB*, 2000.

[6] K. Chakrabarti, M. Ortega-Binderberger, S. Mehrotra, and K. Porkaew. Evaluating refined queries in top-k retrieval systems. 16(2):256–270, 2004.

[7] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker. Making gnutella-like p2p systems scalable. In *ACM SIGCOMM*, 2003.

[8] Y. Chen et al. Quantifying network denial of service: A location service case study. In *Proc. of International Conf on Info. and Comm. Security (ICICS)*, 2001.

[9] E. Cohen, A. Fiat, and H. Kaplan. Associative Search in Peer to Peer Networks: Harnessing Latent Semantics. In *IEEE INFOCOM'03*, April 2003.

[10] E. Cohen and S. Shenker. Replication Strategies in Unstructured Peer-to-Peer Networks. In *ACM SIGCOMM'02*, 2002.

[11] B. Cooper et al. A fast index for semistructured data. In *VLDB*, 2001.

[12] A. Crespo and H. García-Molina. Routing Indices for Peer-to-peer Systems. In *ICDCS'02*, July 2002.

[13] F. M. Cuenca-Acuna and T. D. Nguyen. Text-Based Content Search and Retrieval in ad hoc P2P Communities. In *the International Workshop on Peer-to-Peer Computing*, May 2002.

[14] S. C. Deerwester et al. Indexing by latent semantic analysis. *Journal of American Society of Information Science*, 41(6):391–407, 1990.

[15] P. Druschel and A. Rowstron. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. Submission to ACM SIGCOMM, 2001.

[16] R. Duda and P. Hart. *Pattern Classification and Scene Analysis*. Wiley, New York, 1973.

[17] D. A. Evans and R. G. Lefferts. Design and evaluation of the CLARIT TREC-2 system. In D. Harman, editor, *Proceedings of the Second Text REtrieval Conference (TREC-2)*, pages 137–150, 1994.

[18] C. Faloutsos and K.-I. Lin. Fastmap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In *SIGMOD*, pages 163–174, 1995.

[19] V. Gaede and O. Gunther. Multidimensional Access Methods. *ACM Computing Surveys*, 1998.

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

[21] T. Hofmann. Probabilistic latent semantic indexing. In *Proceedings of ACM SIGIR'99*, pages 50–57, 1999.

[22] L. Jin, N. Koudas, and C. Li. Nnh: Improving performance of nearest-neighbor searches using histograms. In *EDBT*, 2004.

[23] G. Karypis and E.-H. S. Han. Concept indexing a fast dimensionality reduction algorithm with applications to document retrieval and categorization. In *International Conference on Information and Knowledge Management (CIKM)*, 2000.

[24] J. B. Kruskal and M. Wish. *Multidimensional Scaling*. Sage Piblications, Beverly Hills, CA, 1978.

[25] D. D. Lewis. Representation and learning in information retrieval. Technical Report 91-93, Univ. of Massachusetts, 1992.

[26] C. Li, E. Chang, H. Garcia-Molina, and G. Wiederhold. Clustering for approximate similarity search in high-dimensional spaces. *IEEE Transaction on Knowledge and Data Engineering (TKDE)*, 14(4):792–808, 2002.

[27] J. Li, B. T. Loo, J. Hellerstein, F. Kaashoek, D. R. Karger, and R. Morris. On the Feasibility of Peer-to-Peer Web Indexing and Search. In *IPTPS'03*, February 2003.

[28] Q. Li and B. Moon. Indexing and querying XML data for regular path expressions. In *The VLDB Journal*, pages 361–370, 2001.

[29] P. Lyman and H. R. Varian. How much information, 2000. Retrieved from http://www.sims.berkeley.edu/how-much-info on Oct. 2002.

[30] L. Ma, J. Shepherd, and A. Nguyen. Document classification via structure synopses. In *Proc. of the Australasian database conference on Database technologies 2003*, pages 59–65, 2003.

[31] D.-A. Manolescu. Feature extraction—A pattern for information retrieval. In *Proc. 5th Pattern Languages of Programming*, Monticello, IL, 1998.

[32] National Electrical Manufacturer's Association. *Digital Imaging and Communication in Medicine (DICOM)*. NEMA, 2001.

[33] M. Ortega-Binderberger, K. Chakrabarti, and S. Mehrotra. An approach to integrating query refinement in SQL. In *Extending Database Technology*, pages 15–33, 2002.

[34] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project, 1998.

[35] PlanetLab. http://www.planet-lab.org/.

[36] C. D. Prete, J. T. McArthur, R. L. Villars, I. L. Nathan Redmond, and D. Reinsel. Industry developments and models, Disruptive Innovation in Enterpr ise Computing: storage. *IDC*, February 2003.

[37] S. Ratnasamy et al. A scalable content-addressable network. to appear in Proceeding of ACM SIGCOMM, 2001.

[38] G. Salton and C. Buckley. Improving retrieval performance by relevance feedback. *Journal of the American Society for Information Science*, 44(4):288–297, 1990.

[39] H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison Wesley, June 1990.

[40] S. Shi, G. Yang, D. Wang, J. Yu, S. Qu, and M. Chen. Making Peer-to-Peer Keyword Searching Feasible Using Multi-level Partitioning. In *IPTPS'04*, 2004.

[41] K. Sparck Jones and P. Willett, editors. *Readings in Information Retrieval*. Morgan Kaufmann Publishers, 1997.

[42] I. Stoica et al. Chord: A scalable peer-to-peer lookup service for Internet applications. to appear in Proceedings of ACM SIGCOMM, 2001.

[43] C. Tang and S. Dwarkadas. Hybrid global-local indexing for efficient peer-to-peer information retrieval. In *USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI)*, March 2004.

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

[45] J. Xu and W. Croft. Query expansion using local and global document analysis. In *Proceedings of the SIGIR'96*, pages 4–11, 1996.

[46] C. Zhai and J. Lafferty. Model-based feedback in the KL-divergence retrieval model. In *Tenth International Conference on Information and Knowledge Management (CIKM 2001)*, pages 403–410, 2001.

[47] B. Y. Zhao et al. Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, 2003.