A Two-Stage Clustering Method for Semantic Indexing in Peer-to-Peer Information Retrieval

BLAIR HEUER

Northwesten University

TAO TAO

Uinversity of Illinois at Urbana-Champaign

YAN CHEN

Northwesten University

ZHICHEN XU

Yahoo Inc.

and CHENGXIANG ZHAI

Uinversity of Illinois at Urbana-Champaign

Author's address: B. Heuer and Y. Chen, Department of Computer Science, Northwesten University, Evanston, IL 60201. Email: {bheuer,ychen}@cs.northwestern.edu. T. Tao and C. Zhai, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801. Email: {taotao, czhai}@cs.uiuc.edu. Z. Xu, Yahoo Inc., Sunnyvale, CA 94089, Email: zhichen@yahoo-inc.com.

A new paradigm, peer-to-peer (P2P) information retrieval, has attracted much attention recently. One major challenge in P2P information retrieval is to index documents appropriately so as to allow a user to retrieve as many relevant documents as possible through visiting a minimum number of nodes on the network. In this paper, we propose a novel two-stage clustering approach for P2P indexing, which represents an extension of a previous work (pSearch) in two aspects: (1) In the first stage, we use regular document clustering to replace the SVD dimension reduction method, which not only increases efficiency but also clusters the topically similar documents together in a way that better matches user queries. (2) In the second stage, rather than partitioning the document vectors randomly according to the singular values to fit to distributed hash tables (DHT), we group related concepts so that the indices of similar documents are placed together or near to each other, increasing the chance of seeing more relevant documents in the same node. Our experimental results show that grouping related concepts (in the second phase) improves both the performance of SVD-based and clustering-based methods. In terms of identifying concepts, SVD and clustering result in comparable query accuracy, but clustering can be significantly more scalable than SVD. Overall, our system significantly improves the search accuracy and efficiency of pSearch and enables document dynamics in an incremental and efficient manner.

Categories and Subject Descriptors: H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*search process*

General Terms: Algorithms

Additional Key Words and Phrases: Peer-to-Peer System, Information Retrieval, Overlay Network

1. INTRODUCTION

In recent years we have been witnessing explosive growth in the volume of information [Prete et al. 2003]. According to a recent study at University of California, Berkeley [Lyman and Varian 2000], 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. 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 must periodically crawl the entire web to update the index. Such an update 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 their centralized architecture. Another problem with the existing search engines is that they suffer from single-point-of-failure — when some critical centralized components fail, it affects the availability of the whole system.

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 exploits high-speed networks to perform search at every individual node in a network, leading to a potentially more scalable search method.

Indexing of documents on a P2P infrastructure, however, has not been studied sufficiently in the past. Most of the existing works in P2P information retrieval use term-index based approaches [Li et al. 2003; Tang and Dwarkadas 2004] 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 [Li et al. 2003; Tang and Dwarkadas 2004; Shi et al. 2004].

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) [Ratnasamy et al. 2001] for information retrieval on a P2P network [Tang et al. 2003]. But it has the following problems.

- **Accuracy:** pSearch uses SVD to reduce the dimensionality of the original document vectors, and then partition these vectors naively according to the singular values to fit the dimensionality of the DHT. Our experiments show that the grouping of related concepts identified by SVD can improve the document clustering considerably, resulting in better query accuracy.
- **Efficiency:** The matrix computation used in SVD is expensive by nature. To remedy this effect, Tang et al. [Tang et al. 2004] propose to reduce the input matrix size of SVD by first grouping documents into clusters and treat each cluster as a super document. However, both of these approaches group concepts based on singular values and they fail to group related concepts together. Consequently, a query is usually mapped to multiple such unorganized concept groups, and multiple queries would need to be sent to the underlying DHT for retrieval, which is inefficient for

both network and server resource consumption.

- **Dynamics:** pSearch uses static index generation and therefore requires complete redo for adding documents with new concepts.

In this paper, we present an alternative two stage clustering approach, referred to as TCLI (in short for "Two-stage CLustering for Indexing"), to address the problems above: (i) In the first step, a regular document clustering method is used to identify the concepts in a corpus. (ii) In the second step, these concepts are further clustered such that related concepts fall in the same group with a high probability. These groups have sizes that are similar to the dimensionality of the underlying DHT and are used to index the documents. The justification of this approach comes from the "cluster hypothesis" in retrieval, which says that relevant documents tend to form clusters [van Rijsbergen 1979; Hearst and Pedersen 96].

Our experimental results show that grouping related concepts in the second step improves both the performance of the SVD-based and clustering based methods. With grouping, documents that match a given query are more likely to reside on a small number of nodes in the P2P network. This is evident by examining the distribution of the most relevant documents for the queries that we experiment with. Further, using clustering in the first step generates concepts which have better correlation with the documents and queries. In addition, clustering is significantly more scalable than SVD-based methods. Overall, TCLI significantly improves the search accuracy and efficiency of pSearch and enables document dynamics in an incremental and efficient manner.

This paper makes the following contributions.

- **Semantic locality:** A major extension to pSearch that we made 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 P2P network, which we refer to "semantic locality". As a result, a query can be answered by searching on a small number of nodes, improving both query accuracy and efficiency. This is in contrast to pSearch, which partitions document vectors by taking the contiguous chunks of elements in the vectors.

- **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 amounts of information.
- **Adaptation to document dynamics:** The two-phase indexing enables TCLI to incorporate new documents/concepts incrementally and efficiently without sacrificing search performance.

In the rest of the paper, we first give a brief summary of the background work on pSearch in Section 2. We then present our design and discuss each component of the architecture in Section 3. We discuss how our approach can better respond to document dynamics and system/network failures in Section 4. Finally, we present related work in Section 6 and draw the conclusion in Section 7, respectively.

2. BACKGROUND

In this section, we first introduce two popular models for P2P systems, then give a brief summary of the major related work, pSearch and its components.

2.1 Two P2P System Models

In contrast to the traditional client-server system diagram, in P2P system, each node is both a client and a server. There are two major models to form a P2P network: using a set of uncontrolled end user machines vs. using a set of dedicated infrastructure machines. Many P2P file sharing applications are based on the former, such as eDonkey and KaZaa while the commercial P2P service providers using the latter, such as Akamai for content distribution networks. Basically, the commercial P2P service providers install a set of machines in the data centers at various places. These nodes are usually stable and equipped with good bandwidth to access the Internet. Compared with pure end host based P2P model, this scheme is more costly, but usually provides much better performance and security guarantee.

While our scheme is generic and works for both models, in this paper, we focus on the latter model for a real global IR service provider like Google or Yahoo. We can also adapt our scheme to the end user based P2P model by replicating more indices, *e.g.*, onto the neighbors of designated hosts. This will give us even better failure resilience at the cost of extra index storage/maintenance overhead.

2.2 Background on pSearch

An important challenge in P2P information retrieval is to index documents appropriately so as to allow a user to retrieve as many relevant documents as possible through visiting a minimum number of nodes on the network.

The indexing strategy in pSearch [Tang et al. 2003] takes advantage of distributed hash tables (DHTs) to provide efficient support for advanced retrieval algorithms. pSearch represents documents and queries as vectors (points) in a (semantic) Cartesian space. Given a query as a point in the Cartesian space, the problem of finding the most relevant documents is reduced to locating the document points nearest to the query point. To reduce the dimensionality of the document vectors to fit that of the DHT, documents are first converted to a lower dimension vector representation via Singular Value Decomposition (SVD) and are further partitioned into chunks by grouping neighboring elements in the vectors together. Documents are indexed onto the DHT using the vector chunks as hash keys. Because the hash keys derived from documents preserves their semantics, as a result, documents that are similar in semantics are placed close to each other on the DHT with a high probability. During the search time, a query is also mapped to a low dimension vector as the documents do. The query vectors are then used as hash keys to reach nodes in the P2P documents for matching documents. Documents in such nodes are then scored against the query with any retrieval method.

Next, we describe the main components of pSearch, DHT [Ratnasamy et al. 2001] and SVD [Berry et al. 1999], briefly here.

2.3 Distributed Hash Tables (DHT)

The term "DHT" is a catch-all for a set of schemes sharing certain design goals ([Ratnasamy et al. 2001; Stoica et al. 2001; Zhao et al. 2003; Druschel and Rowstron 2001], *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 TCLI 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`.

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 TCLI system. In addition, DHT provides some very nice properties for distributed systems, such as fault-tolerance, robustness [Ratnasamy et al. 2001; Stoica et al. 2001; Zhao et al. 2003; Druschel and Rowstron 2001], and DoS attack resilience [Chen et al. 2001]. 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 pSearch, the underlying DHT is CAN. CAN partitions a d-dimensional Cartesian space into *zones* and assigns each zone to a node. 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. An example CAN is shown in Figure 1. There are five nodes A-E in the overlay. Each node owns a zone in the Cartesian space.

2.4 SVD

pSearch uses Latent Semantic Indexing (LSI) [Deerwester et al. 1990] to reduce the dimensionality of the document vector. It can also collapse synonyms and reduce noise in documents. LSI is based on Singular Value Decomposition (SVD) [Berry et al. 1999] to transform a high-dimensional term vector (computed from Vector Space Model (VSM))

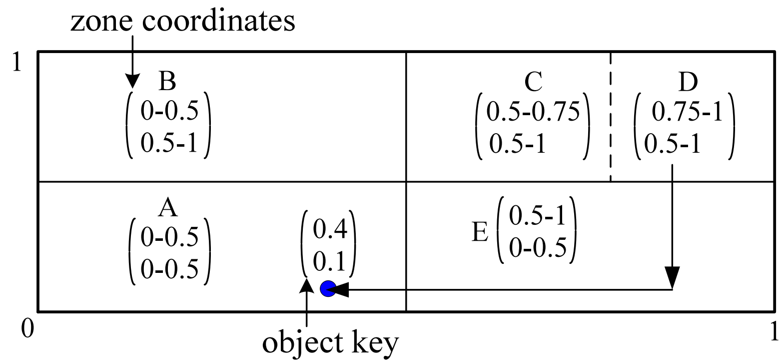

Fig. 1. A 2-dimensional CAN.

into a lower-dimensional *semantic vector*, by projecting the former into a semantic subspace. LSI is capable of bringing together documents that are semantically related even if they do not share terms, by learning from co-occurring word usage. For instance, a search about `car` may return relevant documents that actually use `automobile` in the text.

Let d denote the number of documents in a corpus, and t denote the number of terms in the vocabulary. VSM represents this corpus as a $t \times d$ matrix A, whose entry a_{ij} indicates the importance of term i in document j. Suppose the rank of A is r. SVD decomposes A into the product of three matrices, $A = U\Sigma V^T$, where $U = (u_1, \ldots, u_r)$ is a $t \times r$ matrix, $\Sigma = \text{diag}(\sigma_1, \ldots, \sigma_r)$ is an $r \times r$ diagonal matrix, and $V = (v_1, \ldots, v_r)$ is a $d \times r$ matrix. σ_i's are A's singular values, $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r$.

LSI approximates the matrix A of rank r with a matrix A_l of lower rank l by omitting all but the l largest singular values. Let $U_l = (u_1, \ldots, u_l)$, $\Sigma_l = \text{diag}(\sigma_1, \ldots, \sigma_l)$, and $V_l = (v_1, \ldots, v_l)$.

$$A_l = U_l \Sigma_l V_l^T \tag{1}$$

It is proven that, among all matrices of rank l, A_l approximates A with the smallest error, in the sense of least-squares. The rows of $V_l\Sigma_l$ are the semantic vectors for documents in the corpus. Given U_l, V_l, and Σ_l, the semantic vectors of queries, terms, or documents originally not in A can be generated by folding them into the semantic subspace [Berry et al. 1999].

3. THE ARCHITECTURE AND COMPONENT DESIGN OF TCLI

To address the limits of pSearch as discussed in Section 1, we present an alternative Two-stage CLustering strategy for Indexing (in short, TCLI) in a P2P environment. TCLI extends pSearch in two aspects to achieve better semantic locality: (1) It replaces SVD with more efficient regular document clustering to generate n concepts. This extension has the potential to remove SVD as a bottleneck and enable the system to better respond to document semantics. We refer to this as concept-vector construction. (2) It groups the related concepts with a second-stage clustering to further achieve semantic locality when storing the documents in DHTs. With this second stage clustering, related concepts identified in the concept-vector construction phase are partitioned into groups, with the size of each group less than or equal to the dimensionality of the underlying DHT. The documents are projected into the concepts in each groups to produce so called *index locators*. These index locators are used as DHT keys to map a document onto the DHT. During querying time, queries are mapped into query index locators in the same way and used as DHT keys to identify nodes with matching documents.

Figure 2 illustrates the architecture of TCLI . TCLI is a system built on top of an overlay network — P2P DHT system. Any text document would be placed into this system through the following steps: the concept-vector construction (Section 3.1), index locator construction (Section 3.2), and document placement (Section 3.3). On the other hand, when a query is submitted to TCLI , it is first converted into an index locator, and then a search process

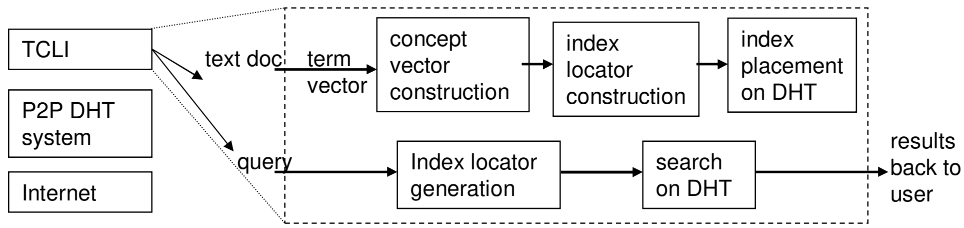

Fig. 2. Layered model and architecture of the TCLI system.

is triggered in the network according to this index locator, and finally retrieval results are generated and sent back to users. This will be detailed in Section 3.4.

3.1 Concept Vector Construction

TCLI indexes documents in two steps. In the first step, the entire document corpus is clustered into clusters with appropriate sizes and density. Rather than relying on SVD which is computationally expensive, we use more efficient concept indexing approach for constructing the concept space [Karypis and Han 2000]. 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.

For document clustering, we 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.

Note that the document clustering only needs to be executed once when we set up the system. Later documents can be added incrementally and in a completely distributed manner as discussed in Section 4.1. To further increase the scalability, we can adopt the distributed clustering techniques such as the Density Based Distributed Clustering as in [Januzaj et al. 2003].

3.2 Index Locator Construction

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. We use DHT (`put`) and (`get`) operations for index insertion and retrieval. In our approach, 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 blindly dividing these concepts into chunks (like pSearch [Tang et al. 2003], which sorts the concepts according to singular values), we group clusters identified in the first step 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. If we randomly divide concept vector into index locators, the related concepts are probably in different chunks. Consequently, a document will need more indices, and indices of related documents tend to be more scattered, causing inefficiency and inaccuracy of querying.

Given the concept groups, 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.

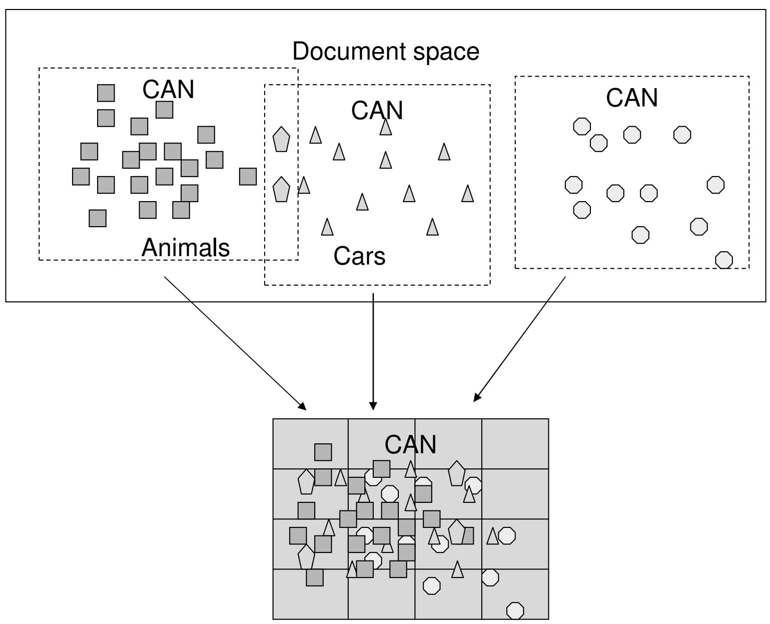

Fig. 3. Document clustering and placement.

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 3 illustrates the basic idea of mapping 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".

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 can compute the axes in a distributed fashion within each peer. For example, we plan to apply the FastMap algorithm [Faloutsos and Lin 1995] 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.

3.3 Index Placement on 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.

3.4 Querying

During query time, a user can contact any node on the DHT overlay. This node is responsible for projecting the query vector onto the concepts of each concept group, to produce a number of index locators. In particular, the following procedure is used 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. Figure 4 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

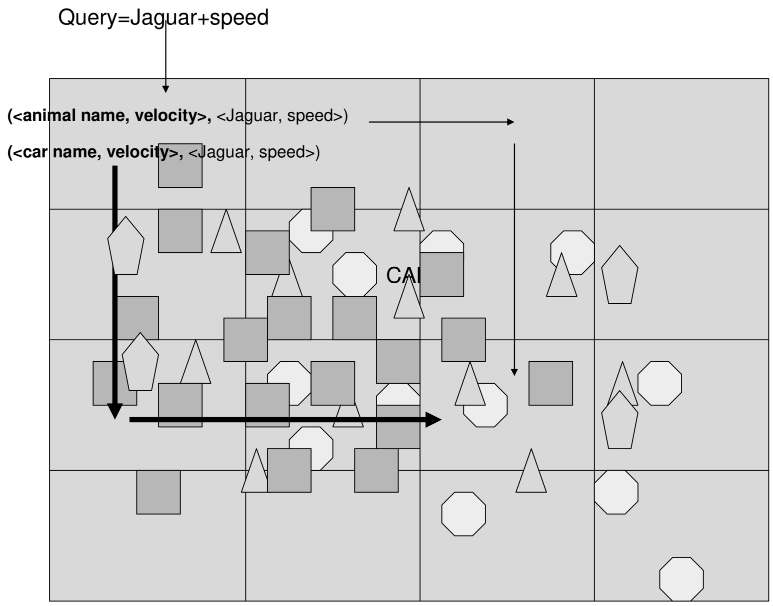

Fig. 4. Query processing.

process identify the matching results of the query in each of the clusters.

For example, in Figure 4, 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.

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 [Ratnasamy et al. 2001]) without flooding. A hypercube geometry treats all concepts equally, whereas other structures (e.g., trees) require matching the concepts in a predetermined order. It is part of our future work to investigate how to enable wildcard search in other types of DHT. In the remaining of the paper, we use CAN as the underlying DHT to build TCLI .

3.5 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 TCLI 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 [Page et al. 1998]. 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.

3.6 Discussion of Generality

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 [Andrzejak and Xu 2002], the random permutation function by Broder et al. [Broder et al. 1998], *etc.*, to eliminate the dependence on the geometry of the underlying logical abstraction of the DHT.

Note that our indexing strategy 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 affects the final retrieval results.

4. SYSTEM PROPERTIES OF TCLI

TCLI is adaptive to document dynamics and is robust to system/network failures. In this section, we discuss these properties.

4.1 Adaptation to Document Dynamics

All the entities in TCLI 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 heavy-weight 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 TCLI nodes, we have a small number of nodes to control new concept generation.

For instance, when new documents on “Bin Laden” 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 Laden” 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 Laden"(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.

4.2 Robustness to System/Network Failures

There are two main reasons why TCLI is robust to system/network failures. First, the DHT can effectively isolate node failures so that even when a subset of nodes fail, the rest can still operate to answer most of the queries. Each node in the DHT exchange `heartbeat` messages with its neighbors. When detecting failures, it will adaptively fix the neighbor tables to route around failures. Thus only the indices on the failed nodes cannot be retrieved. Note that all these monitoring and failure recovery operate in a completely distributed manner.

Secondly, we can further replicate the indices, *e.g.*, to the neighbors of the DHT node where the index locator maps to. Note that the neighbors of a DHT node x are not necessarily in the same subnet or geographically close to x. This will ensure that even when a portion of nodes are disconnected, with high probability, we can still reach all the desired document indices with secondary search.

5. EVALUATION

5.1 Evaluation Methodology

5.1.1 *Experimental Setup.* We implemented a prototype of the TCLI system and evaluated it with two standard TREC data sets (AP88-89 and TREC8). Table I summarizes the statistics of these two data sets.

	#document	avg doc length	queries	#total rel. docs
AP88-89	164597	456.02	TREC 101-150	4805
TREC8	528155	477.32	TREC 401-450	4728

Table I. Experiment data sets

We use Lemur to index the AP corpus. We randomly sampled 25% of the indexed documents to generate a term-by-document matrix. Terms appearing in only one sampled document are not included in the matrix. This leaves us with 41,149 sampled documents and 160,317 index terms in the matrix. We then apply our 2-stage scheme to extract the concepts.

Using this basis, we project all 164,597 documents into the semantic space, computing the index locators for each document. The default values for system setup is in Table II below.

description	default value
number of nodes in the system	1000
# of clusters in stage 1 (# of concepts)	120
dimensionality of CAN (size of index locator)	12

Table II. Default parameters in the evaluation.

5.1.2 *Metrics.* We report all these values in both primary nodes and secondary nodes respectively in the rest of this section. In TCLI , for each query, the nodes which are directly routed to with index locators are primary nodes. We then search the DHT neighbors of the primary nodes, namely secondary nodes. There are two main metrics: accuracy and efficiency.

For accuracy, we use the standard IR metrics, *mean average precision* and precision at 10 and 20 (Pr@10, Pr@20). In addition, given distributed retrieval, we adopt *coverage* which is defined as follows: suppose a query has n relevant documents in the collection. K out of n are in the nodes that a query is directed to (either primary or secondary). Then the coverage is k/n.

The second metric is efficiency, which is measured as the number of visited nodes during a search.

There are some additional metrics used for investigating why our clustering-based concept extraction is better than the SVD-based approach. Given a document/query, we analyze its weight distribution variation when projected onto each of the 120 concepts. Ideally, there are a small number of concepts to capture the document/query with relatively large weight while most of other concepts are not related and have close to zero weight. Such "skewness" is measured with *coefficient of variation* (CV). The CV of a distribution x, defined as below, is a standard metric for measuring inequality of x.

$$CV(x) = \frac{standard\ deviation(x)}{mean(x)} \tag{2}$$

5.2 Evaluation Results

5.2.1 *Accuracy Evaluation and Comparison.* First, we evaluate the accuracy of TCLI , and compare it with other variants as in Table III. The column of "Stage 1" indicates the method used for concept extraction and the column of "Stage 2" shows how to group the concepts into index locators. We further compare them with centralized Indexing and querying with single node (denoted as *centralized*).

Notations	Stage 1: concept extraction	Stage 2: concept grouping
TCLI	clustering	clustering
TCLI+random	clustering	random
pSearch	SVD	sorted by eigenvalues and then divided into equal-size chunks
pSearch+clustering	SVD	clustering
pSearch+random	SVD	random

Table III. Various methods for comparison.

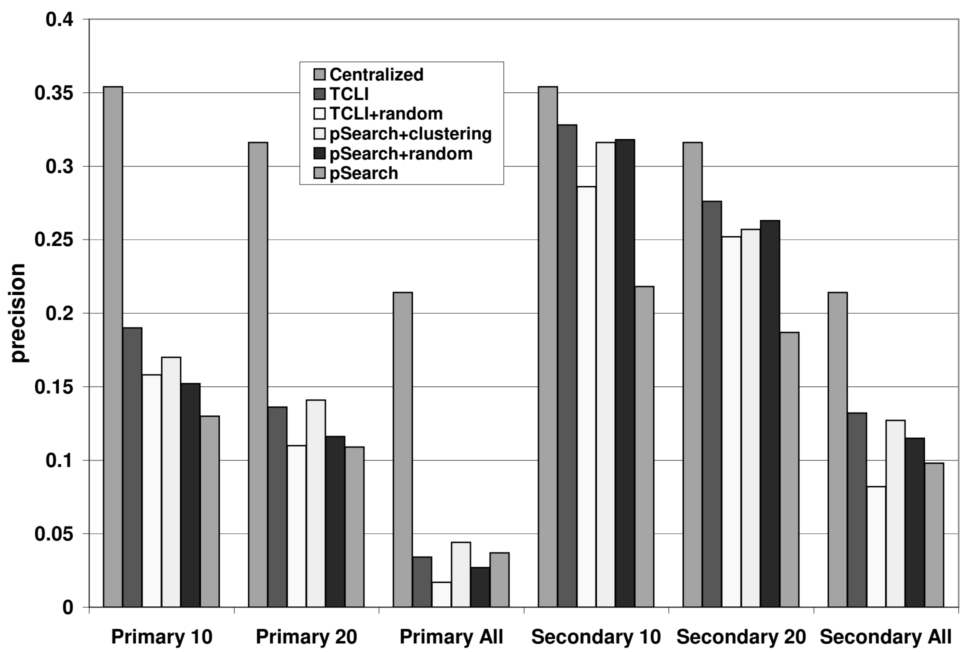

Fig. 5. Precision comparison.

Figure 5 shows the precision comparison results. Centralized approach only has one set of results for precision at 10, 20 or all, which are compared with the primary and secondary results of distributed approaches. The graph shows that the distributed retrieval results on the primary nodes are much worse than those of centralized scheme. But when including the secondary nodes, the distributed retrieval precisions are very close to those of centralized ones.

The figure also shows that TCLI performs better than *TCLI+random*, and *pSearch+clustering* performs much better than *pSearch+random* and *pSearch*. This shows that the second stage clustering significantly improves the results, regardless of the first stage concept extraction method whether it is clustering or through SVD. This indeed confirms with our design principle that *semantic locality (achieved through concept clustering) can significantly improve the search accuracy*.

On the other hand, it seems that the concept extraction method in the first stage can be flexible. The precisions of TCLI are similar, but a bit better than those of *pSearch+clustering* while the precisions of *TCLI+random* are similar but a bit *worse* than to those of *pSearch+random*. Please note that concept extraction with clustering is much more efficient than using SVD, and TCLI overall has the best performance. Thus we recommend using clustering in both stages.

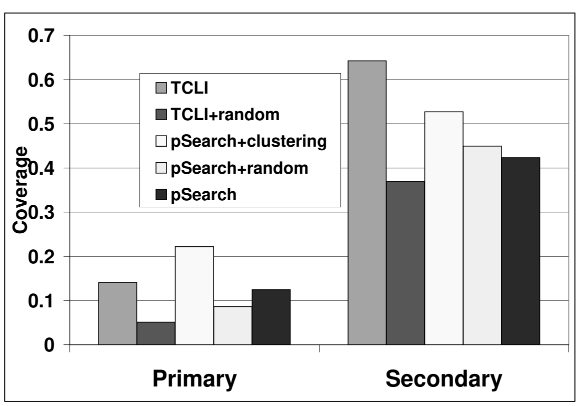

Fig. 6. Coverage comparison.

Figure 6 shows the coverage comparison results. They are similar and consistent to the "primary all" and "secondary all" precision performance in Figure 5.

In addition, our simulation with TREC8 dataset shows similar trend. We omit the graph in the interest of space.

5.2.2 *Effectiveness of Clustering on Each Stage.* To investigate why TCLI outperforms pSearch and its variants, we analyze the effects of clustering on each stage. For the first stage, it is our conjecture that given SVD captures orthogonal concepts with finer granularity, a document, which typically has multiple concepts, gets spread among multiple concepts. Whereas document clustering groups similar documents together, so each document tends to fall into a (major) cluster. Thus the representation of a document would have a more flat vector in SVD than in document clustering. As a result, the relevant documents tend to go into different nodes in CAN because they have high weights on more concepts and thus have a higher chance of getting split into multiple nodes. The fact that SVD spreads a document across multiple concepts is not a disadvantage in many situations. SVD has been shown to be able to handle polysemy and synonymy.

To confirm the conjecture, we first measure that given a document/query, how skewed is its weight distribution with regarding to the 120 concepts extracted via clustering or SVD. We randomly pick 50 queries, the average $CV(clustering) = 4.51$ while the average $CV(SVD) = 0.72$. Furthermore, we randomly pick 50 documents, the the average $CV(clustering) = 1.03$ while the average $CV(SVD) = 0.80$. These results strongly support our conjecture. Interestingly, the contrast is much more significant for queries than for documents.

However, such effect does not show up when comparing the precision of *TCLI+random* and *pSearch+random*. The reason is that without second stage clustering, the effect cannot show up with only 10 index locators. In our experiments, with the first-stage clustering, a query on average has 9.1 concepts with weights over 0.35 (after weight normalization). When these concepts are grouped randomly, it takes on average 9.1 index locators. On the other hand, with concepts generated through SVD in the first stage, a query on average have 25.7 related concepts with weights over 0.35 after weight normalization. Thus when put randomly, it takes all index locators and thus the difference is really small.

The second stage of clustering really helps group related documents together and improves the search accuracy. In our experiment, we analyze the distribution of relevant documents of a query in the nodes of CAN. For each query, we take the relevant documents recorded in the dataset, then analyze its distribution in the DHT nodes. Ideally, these documents reside on a small number of nodes rather than scattered evenly. For each of the indexing methods outlined before, we get an average coefficient of variation for the 50 queries and use it to compare the skewness of document distribution. The results are shown in Figure 7. Again, we observe the similar trend as in Figures 5 and 6. TCLI has the most skewed distribution, followed by *pSearch+clustering*, and the rest three are similar.

5.2.3 *Effects of Parameter Variation.* In this subsection, we vary the parameters to build the TCLI system and test the sensitivity of these parameters.

First, we vary the number of nodes in CAN from 1,000 to 5,000. As shown in Figure 9, although the number of nodes is much larger and documents presumably will be much more scattered. Our TCLI does not degrade much, especially for “primary all” and “secondary all”. On the other hand, we have much better scalability with 5,000 nodes and its performance can be further improved with more index locators (and more concepts when

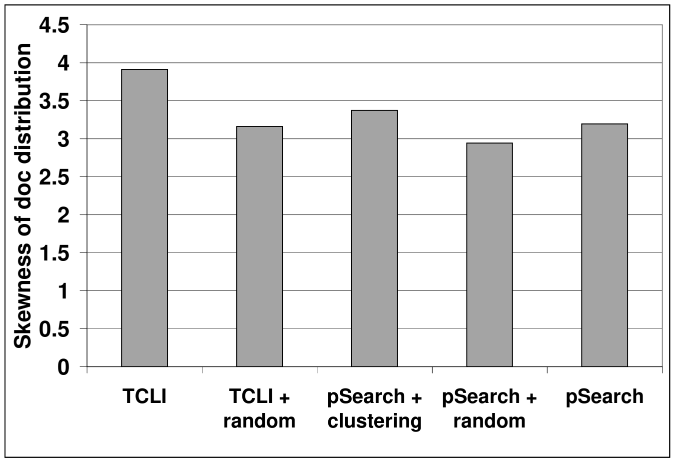

Fig. 7. Skewness of document distribution comparison.

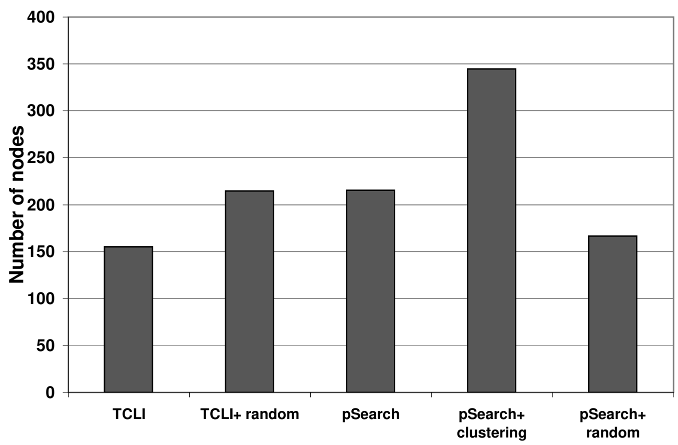

Fig. 8. Efficiency comparison.

applicable) as discussed below.

Secondly, we vary the total number of concepts extracted and consequently vary the number of index locators, and then evaluate how the retrieval accuracy is affected. We use 360 concepts (30 index locators) and compare its precision with 120 concepts (10 index locators) in Figure 10. The retrieval accuracy on primary nodes are better because there are up to 20 more primary nodes. But the performance on secondary nodes are very

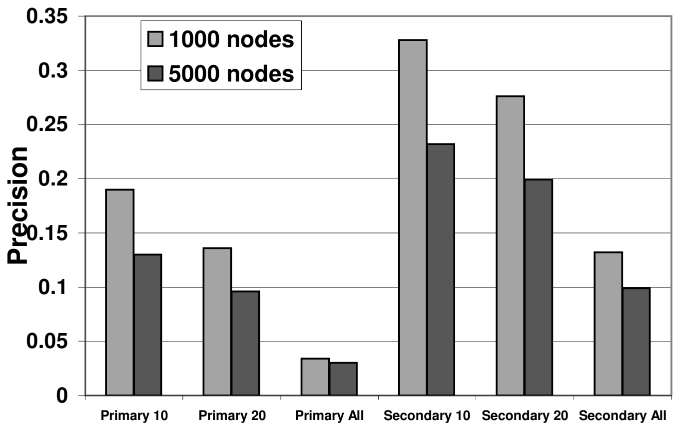

Fig. 9. Precision with varied number of nodes.

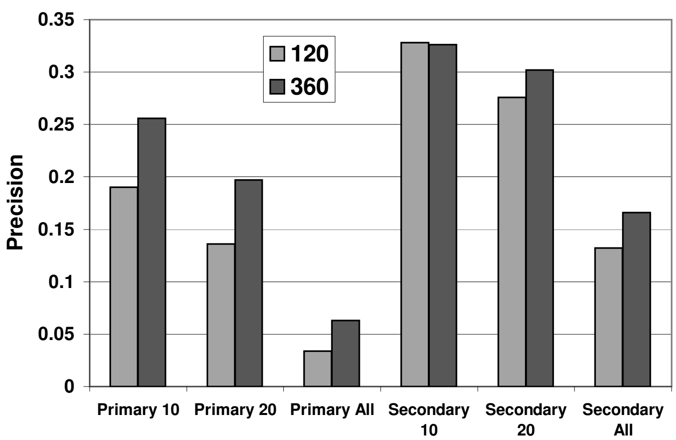

Fig. 10. Precision with varied number of concepts.

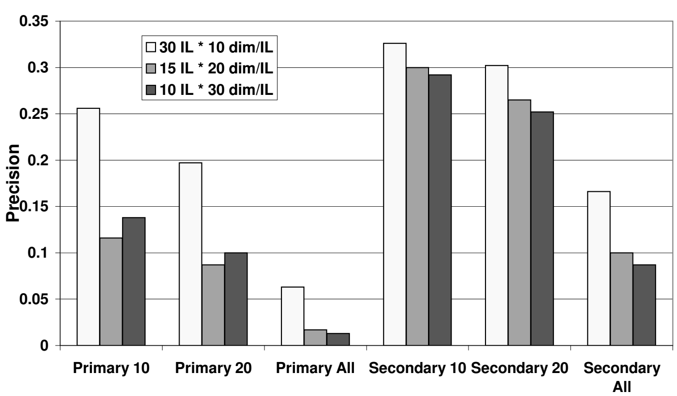

Fig. 11. Tradeoff between the number and the size of index locators (abbreviated as “IL”).

similar, suggesting that with the 1,000-node system, 120 concepts in 10 index locators are sufficient.

Finally, when given a certain number of concepts, we study the tradeoff between the number of index locators and the size of index locators. The results are shown in Figure 11 for a TCLI with 300 concepts. Similarly to the results in Figure 10, 30 index locators with 10 dimension/locator performs the best. But the differences are small for performance including secondary nodes.

In summary, these experiments confirms the second design principles of the TCLI system in Section 1: *TCLI provides flexible tradeoff between search accuracy and efficiency*.

5.2.4 *Search Efficiency.* In this subsection, we evaluate the efficiency of TCLI in terms of the number of nodes traversed. Figure 8 shows the number of secondary nodes visited and TCLI has the least number of nodes visited.

5.2.5 *Search Dynamics.* After we set up the concepts and concept clusters with AP data, we randomly sample 100 documents from TREC8, and use our incremental algorithm to index these documents. We use the same clustering threshold (the maximum cluster diameter) in the static AP dataset to extract concepts. Three new concepts are found and form a new index locator. Querying of these documents returns similar precision results as in previous standalone TREC8 evaluation. We omit the details in the interest of space.

6. RELATED WORK

There has been a huge body of work on building indexing structures to conduct similarly search in a high-dimensional space [Li et al. 2002; Jin et al. 2004; Gaede and Gunther 1998; Samet 1990]. 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 [Kruskal and Wish 1978], Karhunen-Loeve ("K-L") transform [Duda and Hart 1973], Local Dimensionality Reduction [Chakrabarti and Mehrotra 2000], and FastMap [Faloutsos and Lin 1995]. Work on distributed information retrieval (e.g., [Gravano et al. 1999; Callan]) is also related, but a centralized retrieval interface is assumed, which is less robust and less efficient than the proposed TCLI architecture.

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

P2P Query processing has been extensively studied [Cuenca-Acuna and Nguyen 2002; Crespo and García-Molina 2002; Cohen et al. 2003; Chawathe et al. 2003; Cohen and Shenker 2002; Li et al. 2003]. 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 [Sparck Jones and Willett 1997]) and have received much attention in the database community (*e.g.*, [Ortega-Binderberger et al. 2002; Chakrabarti et al. 2004]). Our method 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.

The closest work to ours is pSearch, which uses document indexing and reduces dimensionality of document vectors to produce keys to co-locate topically similar documents on the P2P overlay. This scheme, while elegant, unfortunately, has three major limitations. (1) In pSearch, dimension reduction is performed via SVD followed by subdividing the SVD-generated vectors into subvectors. These subvectors do not correspond to the nature document clusters in the corpus. Consequently, this can cause related terms or concepts to span multiple subvectors, reducing the efficiency and effectiveness of document search. (2) Static index generation requires complete redo for adding documents with new concepts. (3) LSI-based index locator generation is not scalable.

Tang et al. [Tang et al. 2004] improves upon the LSI used in pSearch by first clustering documents into clusters and treat each cluster as a super document in SVD computation. This approach significantly reduces the size of input matrix to SVD and are several order of magnitude more efficient. The paper further describes normalization schemes of the semantic vectors to compensate noises introduced by document clustering. This kind of clustering is in the same spirit to clustering used in the first phase of our approach, same as pSearch, this work does not attempt to group related concepts into groups for document indexing. As shown in our experiments, grouping concepts according to sigular value can produce clusters that are less coherent than random grouping, ad can result suboptial query performance.

7. CONCLUSION

In this paper, we extend a state-of-the-art P2P indexing method (i.e., pSearch) and propose TCLI , 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, as verified in our simulation with AP and TREC data. The TCLI can potentially support query refinement, feedback and personalized search, which can all help to improve the usability.

REFERENCES

ANDRZEJAK, A. AND XU, Z. 2002. Scalable, efficient range queries for grid information services. In *2nd IEEE International Conference on Peer-to-Peer Computing*.

BERRY, M., DRMAC, Z., AND JESSUP, E. 1999. Matrices, Vector Spaces, and Information Retrieval. *SIAM Review 41,* 2, 335–362.

BRODER, A. Z., CHARIKAR, M., FRIEZE, A. M., AND MITZENMACHER, M. 1998. Min-wise independent permutations. In *Proc. of 13th annual ACM symposium on Theory of Computings*.

CALLAN, J. Distributed information retrieval. In *Advances in Information Retrieval*, W. B. Croft, Ed. Kluwer Academic Publishers, 127–150.

CHAKRABARTI, K. AND MEHROTRA, S. 2000. Local dimensionality reduction: A new approach to indexing high dimensional spaces. In *Proc. of VLDB*.

CHAKRABARTI, K., ORTEGA-BINDERBERGER, M., MEHROTRA, S., AND PORKAEW, K. 2004. Evaluating refined queries in top-k retrieval systems. *16,* 2, 256–270.

CHAWATHE, Y., RATNASAMY, S., BRESLAU, L., LANHAM, N., AND SHENKER, S. 2003. Making gnutella-like p2p systems scalable. In *ACM SIGCOMM*.

CHEN, Y. ET AL. 2001. Quantifying network denial of service: A location service case study. In *Proc. of International Conf on Info. and Comm. Security (ICICS)*.

COHEN, E., FIAT, A., AND KAPLAN, H. 2003. Associative Search in Peer to Peer Networks: Harnessing Latent Semantics. In *IEEE INFOCOM'03*.

COHEN, E. AND SHENKER, S. 2002. Replication Strategies in Unstructured Peer-to-Peer Networks. In *SIGCOMM'02*.

CRESPO, A. AND GARCÍA-MOLINA, H. 2002. Routing Indices for Peer-to-peer Systems. In *ICDCS'02*.

CUENCA-ACUNA, F. M. AND NGUYEN, T. D. 2002. Text-Based Content Search and Retrieval in ad hoc P2P Communities. In *the International Workshop on Peer-to-Peer Computing*.

DEERWESTER, S. C. ET AL. 1990. Indexing by latent semantic analysis. *Journal of American Society of Information Science 41,* 6, 391–407.

DRUSCHEL, P. AND ROWSTRON, A. 2001. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. Submission to ACM SIGCOMM.

DUDA, R. AND HART, P. 1973. *Pattern Classification and Scene Analysis*. Wiley, New York.

FALOUTSOS, C. AND LIN, K.-I. 1995. Fastmap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In *SIGMOD*. 163–174.

GAEDE, V. AND GUNTHER, O. 1998. Multidimensional Access Methods. *ACM Computing Surveys*.

GRAVANO, L., GARCÍA-MOLINA, H., AND TOMASIC, A. 1999. GlOSS: text-source discovery over the Internet. *ACM Transactions on Database Systems 24,* 2.

HEARST, M. A. AND PEDERSEN, J. O. 96. Reexamining the cluster hypothesis: Scatter/gather on retrieval results. In *Proceedings of SIGIR'96*.

JANUZAJ, E., KRIEGEL, H., AND PFEIFLE, M. 2003. Towards effective and efficient distributed clustering. In *Proc. of Workshop on Clustering Large Data Sets (ICDM)*.

JIN, L., KOUDAS, N., AND LI, C. 2004. Nnh: Improving performance of nearest-neighbor searches using histograms. In *EDBT*.

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

KRUSKAL, J. B. AND WISH, M. 1978. *Multidimensional Scaling*. Sage Piblications, Beverly Hills, CA.

LI, C., CHANG, E., GARCIA-MOLINA, H., AND WIEDERHOLD, G. 2002. Clustering for approximate similarity search in high-dimensional spaces. *IEEE Transaction on Knowledge and Data Engineering (TKDE) 14,* 4, 792–808.

LI, J., LOO, B. T., HELLERSTEIN, J., KAASHOEK, F., KARGER, D. R., AND MORRIS, R. 2003. On the Feasibility of Peer-to-Peer Web Indexing and Search. In *IPTPS'03*.

LYMAN, P. AND VARIAN, H. R. 2000. How much information. Retrieved from http://www.sims.berkeley.edu/how-much-info on Oct. 2002.

ORTEGA-BINDERBERGER, M., CHAKRABARTI, K., AND MEHROTRA, S. 2002. An approach to integrating query refinement in SQL. In *Extending Database Technology*. 15–33.

PAGE, L., BRIN, S., MOTWANI, R., AND WINOGRAD, T. 1998. The pagerank citation ranking: Bringing order to the web. Tech. rep., Stanford Digital Library Technologies Project.

PRETE, C. D., MCARTHUR, J. T., VILLARS, R. L., NATHAN REDMOND, I. L., AND REINSEL, D. 2003. Industry developments and models, Disruptive Innovation in Enterpr ise Computing: storage. *IDC*.

RATNASAMY, S. ET AL. 2001. A scalable content-addressable network. to appear in Proceeding of ACM SIGCOMM.

SAMET, H. 1990. *The Design and Analysis of Spatial Data Structures*. Addison Wesley.

SHI, S., YANG, G., WANG, D., YU, J., QU, S., AND CHEN, M. 2004. Making Peer-to-Peer Keyword Searching Feasible Using Multi-level Partitioning. In *IPTPS'04*.

SPARCK JONES, K. AND WILLETT, P., Eds. 1997. *Readings in Information Retrieval*. Morgan Kaufmann Publishers.

STOICA, I. ET AL. 2001. Chord: A scalable peer-to-peer lookup service for Internet applications. to appear in Proceedings of ACM SIGCOMM.

TANG, C. AND DWARKADAS, S. 2004. Hybrid global-local indexing for efficient peer-to-peer information retrieval. In *NSDI-04*.

TANG, C., DWARKADAS, S., AND XU, Z. 2004. On scaling latent semantic indexing for large peer-to-peer systems. In *Proceedings of ACM SIGIR*.

TANG, C., XU, Z., AND DWARKADAS, S. 2003. Peer-to-peer information retrieval using self-organizing semantic overlay networks. In *ACM SIGCOMM*.

VAN RIJSBERGEN, C. J. 1979. *Information Retrieval*. Butterworths.

ZHAO, B. Y. ET AL. 2003. Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*.

Received Month Year; revised Month Year; accepted Month Year