Distributed Document Sharing with Text Classification over Content-Addressable Network

Tayfun Elmas and Oznur Ozkasap

Koc University
Department of Computer Engineering
Sariyer, 34450 Istanbul, Turkey
{telmas,oozkasap}@ku.edu.tr
http://www.ku.edu.tr

Abstract. Content-addressable network is a scalable and robust distributed hash table providing distributed applications to store and retrieve information in an efficient manner. We consider design and implementation issues of a document sharing system over a content-addressable overlay network. Improvements and their applicability on a document sharing system are discussed. We describe our system prototype in which a hierarchical text classification approach is proposed as an alternative hash function to decompose dimensionality into lower dimensional realities. Properties of hierarchical document categories are used to obtain probabilistic class labels which also improves searching accuracy.

1 Introduction

With increasing data storage capacity of the Internet, building information retrieval systems that will serve a wide range of purposes would be indispensable. Peer-to-peer (P2P) systems have become a popular alternative to build large-scale platforms for distributed information retrieval applications, as they provide a scalable, fault-tolerant and self-organizing operational environment. Several studies focus on network structures that make use of P2P organizations together with efficient decentralized and localized processing capabilities inherent to the P2P technology [2].

A key point in building P2P systems is organizing nodes in a distributed system to get maximum performance and speed from the overall structure. This can be accomplished by *semantic overlays*, which solve the problem of random distribution of documents among the nodes in the network [1]. With semantic overlays, documents are distributed among the nodes considering their semantic similarities so that the documents similar to each other would be located in a small region.

In addition to semantic issues, an organization scheme is needed to address the nodes, distribute both data and processing over these nodes and access them in an efficient manner. *Content-addressable networks (CAN)* [10] provide distributed hash tables which can be queried using a *key* to get the associated *object*. In the cartesian space, *object* is a point in the space representing queries or documents. Some recent search engines like *pSearch* [13] have used vector space model (VSM) and latent semantic indexing (LSI) that map each document into a term vector building a whole document-term matrix.

P2P networks are becoming increasingly popular since they offer benefits for efficient collaboration, information sharing, and real-time communication in large-scale systems. In such an environment, each peer has a collection of documents which represents the knowledge distributed by the peer. Moreover, each peer shares its information with the rest of the network through its neighbors, namely its *peers*. Each document can be associated with a unique id (e.g., generated by a hash function on the contents of the document) to uniquely identify the same documents located on different peers. A node searches for information by sending query messages to its peers. We can assume that a query is either a collection of keywords or a complete document containing the key information on the topic interested. A peer receiving a query message searches the documents locally among its collection of documents and also propagates the query to its neighbors if specified in searching policy. Most P2P systems use flooding, propagating the query along all neighbors until some threshold criteria is reached. If the search is successful, the peer generates a reply message to the querying peer.

In this study, we propose a distributed document sharing system that is capable of indexing and searching through a collection of documents. The system consists of peers that store indexes to documents and key vectors that identify each document uniquely, with capability of searching through them. Consequently, the amount of data each individual node stores is much less compared to the systems that store the actual contents of documents. In the proposed system, a machine-learning approach [6] is utilized to generate identifiers which uniquely locate a document. We use properties of hierarchical document categories to get probabilistic class (i.e. topic) labels which also improves searching accuracy.

The paper is organized as follows. Operation of CAN in document sharing system and basic protocols are explained in section 2. In section 3, design considerations and improvements are discussed with applications to our proposed system. Section 4 describes the system developed and results obtained. Related work is discussed in section 5. In Section 6, conclusions and future directions are stated.

2 Document Sharing on Content-Addressable Network

Content-addressable network (CAN) proposes a promising approach to organize a P2P overlay network and perform peer operations, transparent to most of the overlay. Each peer knows only a limited number of peers and each operation like joining or leaving can be established in a small fraction of time and in a limited region by not bothering the rest of the network.

CAN is a distributed hash table that maps *key* values to *object* values. *key* is a point in Cartesian space and the corresponding *object* is located in the network node. In a *cartesian coordinate system*, the coordinates of a point are its distances from a set of perpendicular lines that intersect at an origin. The upper limit of the space is specified as 1 in our system, thus a point can have coordinates as floating-point numbers between 0 and 1. In this space, each node has a corresponding *zone* in Cartesian space.

In the original definition, *reality* refers to multiple coordinate spaces with the same dimensionality. Further details on this concept are given in Section 3.2. In this study, we redefine the term reality. We use different dimensions for each reality where each reality corresponds to a topic group in topic hierarchy. In addition, we divide a key point into subkey points and perform routing and search to reach different subkey points of each reality for a single key. Therefore, the CAN space is decomposed into realities according to class hierarchy, and similarities were computed by separate realities for separate parts of class weight vector.

There are some fundamental operations in CAN networks, such as joining as a new node, publishing a document and searching through existing documents. All these operations require routing through the network to reach a desired node. In this section, these basic operations will be introduced considering our new reality definition and subkey concept.

2.1 Routing

Routing from a source node to a destination node on the overlay network is equivalent to routing from the zone of the source node to the destination zone in the Cartesian space, so that a message with key point is passed from the sender, through some other nodes, until it reaches the node whose zone surrounds that key point. The routing process here is established for a subkey in its associated reality. Since each node has direct connection to only its neighbors, a node uses its neighbors to redirect a message. Simple approach uses greedy choice. Once a node receives a message with a key point, it checks if its zone contains the point. If so, it handles the message according to its contents. Otherwise, it computes the distances between its neighbors and the key point, and chooses the neighbor with the smallest distance to the point. Then it passes the message to that neighbor.

2.2 Joining

The CAN space is divided amongst the nodes currently running in the system. To allow the CAN to grow incrementally, a new node that joins must be given its own portion of the coordinate space. This process is called the *join process* and a new node must join all realities defined in the CAN. The phases are as follows:

- A new node that wants to join the CAN space first discovers the IP host address of any node currently in the system. This address can be directly configured for the new node, which is actually done in this study, or can be retrieved through a bootstrapping mechanism as explained in [10].
- The new node randomly chooses a point P in the space and divides it into subkeys for each reality. Then it sends a join request message destined for each subkey of P through the associated reality. Each CAN node then uses

the routing process to forward the message in the reality. The destination node splits its zone in half and assigns one half to the new node. The split is done according to order of dimensions and the order decides along which dimension a zone is to be split, so that zones can remerge when a node leaves the CAN.

– Then there is some information exchange. First, information about the new zone is sent to the new node. After getting its zone in CAN space, the new node learns its neighbors set from the previous occupant. This set is a subset of the previous occupant's neighbors, plus that occupant itself. Similarly, the previous occupant updates its neighbor set to eliminate those nodes that are no longer neighbors. Finally, both the new and old nodes' neighbors are informed of this join operation.

The join of a new node affects only a small number of nodes ($O(d)$ where d is the number of dimensions) in a small region of the coordinate space. The number of neighbors of a node depends only on the dimensionality of the coordinate space, and is independent of the total number of nodes in the system.

2.3 Publishing

Publishing a document into the system requires some preprocessing to get *key* and *value* pair related to document which will be stored in CAN overlay. The original document is kept in the source machine whether this server is inside or outside CAN overlay, only URL address to that document needs to be stored. Once the URL of the document is obtained, accessing the original document is straightforward. Key generation process is performed by the *hierarchical Expectation Maximization (EM) algorithm* [6] that classifies a big number of documents which has no categorization information using a small number of categorized documents. Details of the EM algorithm implemented in this study are given in the Appendix. Once the classification finishes, the resulting class weight vectors are written in a file. The node publishing these documents reads the file and sends *(key,value)* pairs into the CAN overlay. Publishing process is depicted in Figure 1, and can be summarized as follows:

– Since class labels and thus class weights for most of the documents are not known, EM algorithm is used to assign probabilistic class labels to documents. After applying EM algorithm, the class weights along with the URL addresses are written to a file for the publishing application to read.
– The publishing application does not have to be inside the CAN system. It gets the IP address of any running bootstrap node in CAN and sends a message with *(key,value)* pairs. The communication between outside nodes and those in CAN is assumed to be secured.
– Once the inside node gets the *(key,value)* pair, it just sends this pair to the node which owns the zone containing the point represented by vector *key*.

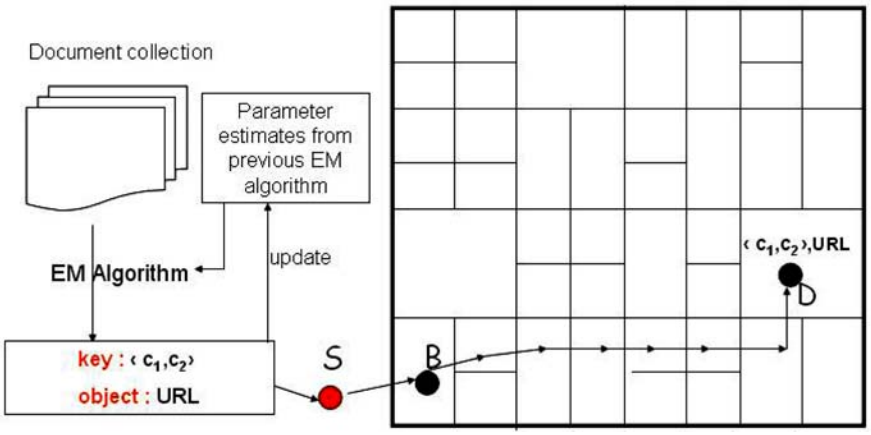

Fig. 1. Publish process for a collection of documents

2.4 Searching

Our system treats any expression, whether a keyword phrase or an ordinary document, given for search operation as the whole query document. Then the naive bayes classification algorithm [6] is used to classify the query document. It uses statistics on previously published documents and generates the vector whose elements are class labels of query document. Since naive bayes uses results of previous results of EM algorithm, it's assumed that most of the documents will be published from the same set of nodes. Naturally, it's common for usenet groups since usenet messages are stored in a fixed set of servers and can be classified on those servers. The search operation is as follows:

– The search operation can be initiated mostly by an outside node. As for publish process, the outside node contacts to a bootstrap node inside CAN and sends search request message with the query key-point.
– Once a node inside CAN gets the query point, it routes the search request message until the corresponding node that owns the surrounding zone is located. That node initiates the actual search process by flooding search messages through its neighbors. When a node gets the search message, it computes the similarity between its data and the query, and sends relevant document indices to the initiator node directly. It also computes the distance between its neighbors and the query point and if the distance does not exceed the threshold, sends the search message to its neighbors. If the document vector is X and the query vector is Y then the similarity measure is the cosine of the angle between these vectors as follows:

$$Cosine(X,Y) = \frac{X \odot Y}{|X| \cdot |Y|} = \sum_{i=1}^{l} X_i Y_i$$

– When a node determines that the distance from the query point exceeds the threshold, it stops the flooding process. The initiator node collects the results and delivers it to the user in a sorted format.

3 Improvements and Discussion

There are several improvements that contribute to the performance of operations on CAN overlays [10]. In this section, we discuss the set of improvements utilized in our prototype system implementation.

1. ***Multidimensional Coordinate Space:*** The system design does not put a limit on the dimensionality of the coordinate space. In fact, increasing dimensions of the CAN coordinate space reduces the routing path length, the number of nodes a message will pass through before reaching its destination. This in turn reduces the path latency only for a small increase in the size of the coordinate routing table of individual nodes. With this feature, since a node has more candidate next hop nodes, fault tolerance in routing mechanism also improves especially in the case of some neighbor node crashes.
   In the classification scheme, we divide the class vectors into sub-vectors according to the class hierarchy, so the number of document classes in a branch of the topic hierarchy determines the dimensionality of its reality. An example is given in Figure 2. As the topic hierarchy grows and leaf classes increase, the dimensionality of realities and thus the system will increase and the routing mechanism will benefit from this. However, decomposing space according to the topic hierarchy reduces the total dimensionality, and computation cost in search operation, since most of the computation will take place only on a small number of realities.
2. ***Using Realities:*** Multiple coordinate spaces can be maintained, and each node in the system can be assigned to a different zone in each coordinate space which is called *reality*. Thus, if there exists r realities in a CAN, a single node is assigned to r coordinate zones, one on every reality and has r independent neighbor sets. This leads to the replication of information, namely document indices for every reality. The replication improves data availability and fault tolerance of routing process, because if a routing process fails on one of the realities, messages can continue to be routed over the other realities. Since the contents of the hash table are replicated on every reality, routing to a key point means reaching the given key on the nearest node on any reality. When forwarding a message, a node examines all its neighbors on each reality and selects the neighbor with coordinates closest to the destination.
   Hierarchical structure of classes makes weights of classes with common ancestors to be closer to each other facilitating to divide the whole space into lower dimensional realities, each reality representing an independent virtual space. If the class weight vectors resulting from classification are used as the only hash function, then those vectors are decomposed into lower dimension vectors. Each of these is associated with a common ancestor, so that class labels in vectors are close to each other. Then each reality represents a branch in hierarchy. This fact reduces computation and latency in parallel executions, since most of the computation will take place in realities associated with common branches.

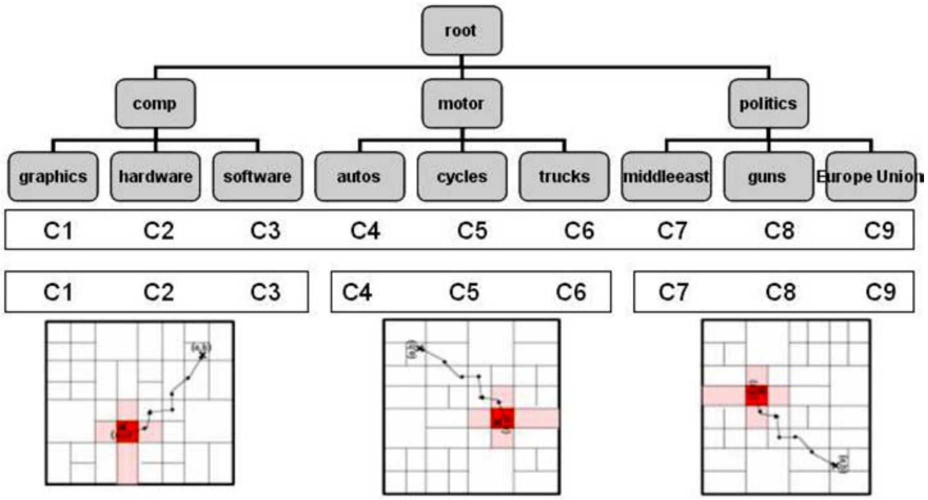

Fig. 2. Decomposing a class vector to sub-vectors

3. ***Multiple Hash Functions:*** Different hash functions can be used to map a single document onto different points in the coordinate space and accordingly replicate the data at distinct nodes in the system. Then, queries for a particular hash table entry could be distributed to all nodes in parallel, reducing the average query latency. This is possible at the cost of increasing the storage size and query traffic. Text classification brings an approach which uses leaf classes of a class hierarchy as dimensions of a space bounding from 0 to 1, and represents a document with probabilistic class labels in that space. Without decomposing whole class vectors to realities, classification can only be used as an additional hash function along with LSI since resulting vectors only cover a limited space. Note that the region comprising the classified documents will be $P(c_1|d_i) + P(c_2|d_i) + \cdots P(c_j|d_i) = 1$, where $P(c_j|d_i)$ is the probability of *class j* given *document i*. It can be used to improve searching performance since the zones to distribute a query can be easily determined and sent to only a small fraction of all neighbors of a node.

4 Prototype and Results

To see the effect of text classification on CAN performance, a prototype document sharing system was implemented in Java. In our system, each peer is capable of joining an existing CAN overlay, publishing documents and searching through them. The hierarchical EM algorithm was implemented as the classification module to generate key vectors for documents. The module writes its output into a file for peers to read and publish into the CAN. Using previous classifications, it also classifies query documents and generates query-key points to send as search message.

For evaluating classification performance of the algorithm, two collections of data were prepared. All documents and vocabulary were selected arbitrarily and without any assumption. No preprocessing were applied to documents. One of the test data sets consists of 8 leaf classes and an hierarchy of 3 levels. 4044 unlabeled and 576 labeled documents were downloaded from Microsoft's news site[1] and 828 of them were separated as unlabeled heldout data to update shrinkage weights. The vocabulary was build with 26411 words. Second test data set consists of 19 leaf classes and an hierarchy of up to 5 levels. 15658 unlabeled and 4788 labeled documents were downloaded from Microsoft's news site and Tom Mitchell's site[2], and 3420 of them were separated as unlabeled heldout data to update shrinkage weights. The vocabulary was build with 27907 words.

Some documents are exchanged between labeled-unlabeled-heldout collections, and results are collected on average. The algorithm converged at an accuracy of 30% for the first test set and at accuracy of 42% for the second test set. The basic EM algorithm used for the first data set converged at an accuracy of 20%. Note that increasing the number of labeled documents improves the performance. Number of labeled documents in the second data set was doubled and an accuracy of 55% was observed.

Some documents were selected arbitrarily and applied to get query key vectors. These query vectors were sent into CAN overlay and returned document indices were evaluated, detecting that the search latency and cost for computing similarities reduce compared to the results with only one reality. It can be stated that using class labels from classification of documents can be used efficiently along with proven techniques like LSI and VSM [3] as a hash function. Also, the closeness of labels with the existence of class hierarchy can be used to divide the overall CAN space into realities.

The documents stored in the system are collected from usenet news groups so that there is a hierarchy in which leafs of the hierarchy become the concrete classes. The hierarchical organization of documents relates documents in different leaf classes so that class labels of related classes with common ancestors will converge each other. This method deals with curse of dimensionality that appears in situations where documents are represented with high dimensionality vectors. If the classifier assigns class labels to documents correctly there will be no circumstance in which labels of classes that are unrelated and far from each other in dimensionality converge to each other.

There is a drawback that this approach classifies documents as a whole and does not take their individual parts in content into consideration, which LSI does. In effect, similarity between two documents is measured by the closeness of them to some topics, not by similarities of their contents to each other. Therefore, a big number of leaf classes are required to measure the similarity as relevant as the user expects. Nevertheless, the topic hierarchy in nowadays' usenet groups are extensive enough to differentiate two documents according to their topics.

[1] news.microsoft.com

[2] www-2.cs.cmu.edu/afs/cs/project/theo-11/www/naive-bayes.html

5 Related Work

pSearch system [13] is one of the distributed information retrieval systems that makes use of a variant of CAN called eCAN, which provides the semantic overlay. The system uses Latent semantic indexing (LSI) [3] to map document to semantic vectors in the space. LSI uses singular value decomposition, and includes semantic concerns like synonyms and uses conceptual vectors instead of single term weights. In vector space model (VSM) [5], the documents and queries are represented as vectors each of whose elements is the weight of the term in the document or the query. VSM uses the product of the frequency of the term and inverse document frequency.

In [8], a novel strategy for clustering peers that share similar properties, forming additional attractive links over the existing network to group similar peers, is proposed. In order to make use of clustered P2P network efficiently, a new query routing strategy called the Firework Query Model, is introduced, which aims to route the query intelligently to reduce the network traffic caused by query passing in the network.

Crespo and Molina [1] propose the semantic overlay network (SON) concept in which nodes are grouped by semantic relationships of documents they store. All nodes store information about the classification hierarchies, and routed queries accordingly. SON concept is devised especially for the creation of a network structure that improves query performance with flexibility and high node autonomy. They also propose creation of multiple overlay networks, in a P2P system, to improve search performance. They ignore the link structure within an overlay network and represented an overlay network just by the set of nodes in it.

Another study introduces Passive Distributed Indexing (PDI) [4], which is a general-purpose distributed search service for document exchange for mobile applications, and based on P2P technology. PDI defines a set of messages for transmission of queries and responses, and all those messages are exchanged using local broadcast transmission. PDI is intended to provide a general-purpose file search service to be used by different types of upper layer applications.

Similar to [13, 8, 1], our approach adopts the similarity relationship among the documents in the system in order to form a cartesian-space abstraction of the content distribution space. In contrast to these studies, we make use of a text classification technique and probabilistic closeness of documents to some classes to obtain the similarity information. We then map those information to a CAN overlay to access desired content efficiently.

Nakao, Peterson and Bavier [7] discuss that one common characteristic of overlay services is implementation of an application-specific routing strategy and propose a new architectural element, a routing underlay, that sits between overlay networks and the underlying Internet. Overlay networks query the routing underlay, which extracts and aggregates topology information from the underlying network, when making application-specific routing decisions. Their approach can be adapted to content-based overlays, like CAN, to reduce the tradeoff between the high-level content-based organization and the network layer topology of an overlay.

The Plaxton data structure [9], which is called a Plaxton mesh is novel that it allows messages to locate objects and route to them across an arbitrarily-sized network, while using a small constant-sized routing map at each hop. Additionally, it guarantees a delivery time within a small factor of the optimal delivery time, from any point in the network. The system makes the assumption that the Plaxton mesh is a static data structure, without node or object insertions and deletions. Objects and nodes have names independent of their location and semantic properties.

Pastry [11] is a scalable, fault resilient, and self-organizing P2P infrastructure. Each node in Pastry has a unique, uniform randomly assigned id in a circular 128-bit identifier space. Given a 128-bit key, Pastry routes an associated message towards the live node whose id is numerically closest to the key. Moreover, each Pastry node keeps track of its neighboring nodes in the namespace and notifies applications of changes in the neighbor set. Each node also maintains a leaf set, which ensures reliable message delivery and is used to store replicas of application objects.

Tapestry [14] provides location-independent routing of messages directly to the closest copy of an object or service using only point-to-point links and without centralized management. Similar to CAN architecture, Tapestry uses randomness to achieve both load distribution and routing locality. It is very similar to Pastry but differs in its approach to mapping keys to nodes in the sparsely populated id space, and how it manages replication. In Tapestry, there is no leaf set, and neighboring nodes in the namespace are not aware of each other.

Like Pastry, Chord uses a circular id space [12]. Unlike Pastry, Chord forwards messages only in clockwise direction in the circular id space. Instead of the prefix-based routing table in Pastry, Chord nodes maintain a finger table, consisting of pointers to other live nodes. Each node also maintains pointers to its predecessor and to its successors in the id space, which is called the successor list.

Each of those infrastructures [9, 11, 14, 12] for building and accessing overlays aim to exploit a structural organization in order to access desired nodes efficiently. CAN overlays have been devised for similar purposes, and they have also the advantage of providing a content-based organization. Moreover, they can fit well to content distribution frameworks to be leveraged in various operations.

6 Conclusions and Future Work

We demonstrate the basic aspects of a distributed document sharing system utilizing text classification techniques over a CAN, and the resultant prototype is assessed to be promising one to build detailed information retrieval systems. It favors availability of both data and processing by providing zone replication and fault-tolerant routing mechanism. In addition, the proposed text classification scheme to generate key vectors for documents provides an additional hash function that reduces the computation in search operation, limiting the search space.

It can also be used to decompose the key vector into lower dimensional vectors, separating overall space into multiple spaces.

There are some issues to be dealt with especially in search operation to make use of the reality decomposition as efficient as possible. In the existence of a huge hierarchical topic hierarchy, text classification would perform better, as long as optimal parameters like search and distance thresholds are supplied to the system. Another issue is constructing the overlay considering the underlying topology to have a better intuition on the effect of CAN message delivery mechanism.

Appendix

The EM algorithm [6] used in this study can be summarized as follows:

- Separate all data into three parts, *labeled, unlabeled and heldout* collections. Note that number of unlabeled documents are much bigger proportional to others.
- Prepare a vocabulary containing all words with some pruning irrelevant ones.
- Specify the class hierarchy with concrete classes at leaves. The hierarchy is defined within a configuration file and it's fed into the algorithm at the beginning.
- Build a naive bayes estimator at each leaf node of the class hierarchy using only labeled data owned by the leaf. Then, form the initial global estimator using the naive bayes estimators built at leaves.
- Classify all unlabeled documents using this naive bayes classifier built previously and place each unlabeled document into the document collection of resulting leaf class $argmax(c_j)$. This step can be considered as the *first E step* in EM algorithm.
- Iterate over the entire class hierarchy, until log likelihood of the global estimate, $\log P(\theta|D)$, converges:
  - **M Step:** Compute a naive bayes estimates at each node of the hierarchy starting from leaf nodes and going through until root. Documents used in an estimation at an ancestor node of c_j comprise documents owned by its children, except those used previously for the same class c_j. After all estimates finish, construct the global estimate using weighted sum of the estimates for each leaf class along the path from the leaf to the root.
  - **Weight Update:** Classify all heldout documents using the global classifier, and using the results of the classifications update weight of each node with respect to all leaf classes.
  - **E Step:** Using the global naive bayes estimator, reclassify all unlabeled documents and replace those whose resulting class changes.
- After finishing with EM algorithm, record the class vectors of documents consisting probabilistic class labels to a file for the publisher node to read and send into the overlay. In addition, record the parameters of the last global estimator to classify a given query document to get a key vector for searching later.

References

1. Arturo Crespo and Hector Garcia-Molina. Semantic overlay networks for p2p systems. Technical report, Computer Science Department, Stanford University, October 2002.
2. Zeinalipour-Yazti D. Information retrieval in peer-to-peer systems. *Master Thesis, Department of Computer Science University of California*, 2003.
3. Scott C. Deerwester, Susan T. Dumais, Thomas K. Landauer, George W. Furnas, and Richard A. Harshman. Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, 41(6):391–407, 1990.
4. Christoph Lindemann and Oliver P. Waldhorst. A distributed search service for peer-to-peer file sharing in mobile applications. In *Proceedings of the Second International Conference on Peer-to-Peer Computing*, page 73. IEEE Computer Society, 2002.
5. Z. Drmac M. Berry and E. Jessup. Matrices, vector spaces and information retrieval. *SIAM Review*, 41(2):335–362, 1999.
6. A. McCallum and K. Nigam. Text classification by bootstrapping with keywords, em and shrinkage. In *ACL Workshop for Unsupervised Learning in Natural Language Processing*, 1999.
7. Akihiro Nakao, Larry Peterson, and Andy Bavier. A Routing Underlay for Overlay Networks. In *Proceedings of the ACM SIGCOMM Conference*, August 2003.
8. Cheuk-Hang Ng, Ka-Cheug Sia, and Irwing King. A novel strategy for information retrieval in the peer-to-peer network.
9. C. Greg Plaxton, Rajmohan Rajaraman, and Andrea W. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *ACM Symposium on Parallel Algorithms and Architectures*, pages 311–320, 1997.
10. Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content addressable network. In *Proceedings of ACM SIGCOMM 2001*, 2001.
11. Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Lecture Notes in Computer Science*, 2218:329–350, 2001.
12. Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of ACM SIGCOMM*, pages 149–160. ACM Press, 2001.
13. Chunqiang Tang, Zhichen Xu, and Sandhya Dwarkadas. Peer-to-peer information retrieval using self-organizing semantic overlay networks. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 175–186. ACM Press, 2003.
14. B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, UC Berkeley, April 2001.