RC23145 (W0403-068) March 9, 2004
Computer Science

IBM Research Report

Supporting Efficient Keyword-Based File Search in Peer-to-Peer File Sharing Systems

Lintao Liu, Kyung Dong Ryu
Department of Computer Science & Engineering
Arizona State University
Tempe, AZ 85287

Kang-Won Lee
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

Supporting Efficient Keyword-based File Search in Peer-to-Peer File Sharing Systems

Lintao Liu **Kyung Dong Ryu** **Kang-Won Lee**

Dept. of Computer Science & Engineering
Arizona State University
Tempe, AZ 85287, USA
{lintao.liu, kdryu}@asu.edu

IBM T.J. Watson Research Center
Hawthorne, NY 10532, USA
kangwon@us.ibm.com

***Abstract*— Peer-to-Peer (P2P) computing has become a popular distributed computing paradigm thanks to abundant computing power of modern desktop workstations and widely available network connectivity via the Internet. Although P2P file sharing provides a scalable alternative to conventional server-based approaches, providing efficient file search in a large scale dynamic P2P system remains a challenging problem. In this paper, we propose a set of mechanisms to provide a scalable keyword-based file search in DHT-based P2P systems. Our proposed architecture, called *Keyword Fusion,* balances unfair storage consumptions at peers, transforms users' queries to contain focused search terms. Through trace-driven simulations, we show that Keyword Fusion can reduces the storage consumption of the top 5% most loaded nodes by 50% and decrease the search traffic by up to 67% even in a modest scenario of combining two keywords.**

***Keywords*— P2P file sharing, keyword-based file search, distributed hash table (DHT), inverted distributed hash table**

I. INTRODUCTION

Peer-to-Peer (P2P) computing has become a popular distributed computing paradigm thanks to abundant computing power of modern desktop workstations and widely available network connectivity via the Internet. Although P2P file sharing provides a scalable alternative to conventional server-based approaches, providing efficient and robust file search in P2P systems has been a key challenge. To provide effective search for desired files in large-scale P2P networks, distributed hash table (DHT) has been proposed [1-4]. In DHT-based P2P systems, locating a node that contains a particular file is simply done by querying a distributed lookup table that stores *<file ID, value>* mapping over multiple DHT nodes, where *file ID* denotes the globally unique ID of the file and the *value* represents the location of the file. These DHT-based approaches guarantee efficient discovery of an existing file in a small bounded number of network hops (O (log N)) for a network consisting of N nodes.

Although DHTs provide efficient lookup service, files can be located only through their globally unique IDs. Oftentimes, however, users may wish to search for files using a set of descriptive keywords or do not have the exact ID of the files. To provide a search capability using keywords to the users, an extension called the *inverted distributed hash table* method has been proposed [5, 6]. The main idea of the inverted DHT is to use keywords as indices of a DHT to locate files by maintaining *<keyword, list of values>* information at each DHT node, in place of *<file ID, value>*. Note that because the same keyword can appear in multiple files, unlike in the case of file ID, the right hand side of the mapping is extended to store a list of values to include the locations of all files containing that keyword . In addition to facilitating the basic mechanism for keyword-based search, the related work has proposed several optimization techniques to reduce the query traffic using Bloom filter to compress the intermediate results [5, 6] and caching previous results [5].

The main focus of this paper is to address an important challenge, called *the common keyword problem,* inherent to a peer-to-peer file sharing system that employs an inverted hash table mechanism. The common keyword problem arises from the fact that certain keywords are commonly associated with a very large number of files compared to other keywords [7]. Consequently, a small number of peers, which are responsible for storing location information for such *common* keywords, will have to consume excessive amount of storages than other peer nodes. This induces severe unfairness among its users, and therefore it may discourage users from participating in P2P networks. Furthermore, a search query containing these *common* keywords will generate a huge volume of network traffic since they are associated with a very long list of file location information. To make matters worse, not all the search results may be used to answer the user's query since these common keywords tend to be too generic and thus may contain a large amount of irrelevant information.

In this paper, we propose a novel keyword-based file search mechanism called *Keyword Fusion*. This mechanism provides a scalable and efficient solution for DHT-based P2P file sharing systems, where files are annotated with descriptive keywords (as in the case of music, pictures,

video files). In particular, Keyword Fusion is a fully decentralized architecture with the following features:

- Utilizing a distributed data structure, called the *Fusion Dictionary,* which stores certain common keywords in the system, *Keyword Fusion* transforms queries to more specific search terms and thereby improves search efficiency.
- Safely deleting excessively large lists of files containing common keywords or redistributing them over the entire network, *Keyword Fusion* can mitigate the level of unfairness in storage consumption.

Based on a set of distributed algorithms, which can be easily incorporated into an existing DHT-based P2P lookup service, Keyword Fusion offers a low overhead solution to the inefficiency in search and unfairness in peer overhead in current keyword-based search mechanisms. Through trace-driven simulations, we show that Keyword Fusion can reduces the storage consumption of the top 5% most loaded nodes by 50% and decrease the search traffic by up to 67% even in a modest scenario of combining two keywords. Our proposed mechanism is designed for searching multimedia files annotated with related keywords, such as images and movies, rather than full-text searches of documents.

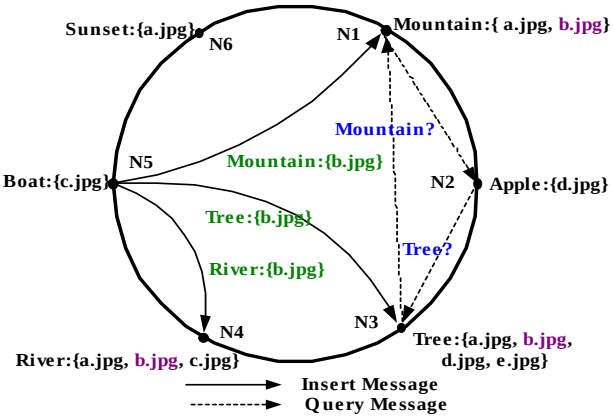
The rest of this paper is organized as follows. Section II provides a brief overview of the inverted hash table method. Section III introduces the Keyword Fusion architecture and addresses the common keywords problem. Section IV evaluates the effectiveness of Keyword Fusion using an extended Chord simulator and annotated multimedia file data sets. Section V describes related work, and finally, Section VI concludes this paper.

II. KEYWORD SEARCH IN DHT-BASED P2P SYSTEMS

This section presents an overview of the inverted distributed hash table mechanism using Chord [1] as a reference DHT, to support keyword search capability.

To describe briefly, in DHT-based P2P systems, locating a node that contains a particular file is simply done by querying a distributed lookup table that stores *<file ID, value>* mapping, where *file ID* denotes the globally unique ID of the file and the *value* represents the location of the file. To accommodate typically huge ID space, this *<file ID, value>* mapping information is distributed and stored over multiple DHT nodes. When assigning a file ID to a node, consistent hashing is used so that the load is evenly distributed across the entire nodes. To facilitate efficient routing, DHT-based P2P systems organize the participating nodes into a logical overlay structure. In case of Chord, the nodes are organized into a logical ring.

When a node receives a query message, it looks up the local routing table (called "finger table") that contains the next node information in the logical ring and forwards the message to the next node. Chord organizes the finger table



File ID	Keywords
a.jpg	Tree, River, Mountain, Sunset
b.jpg	Tree, River, Mountain
c.jpg	Boat, River
d.jpg	Apple, Tree
e.jpg	Tree

Figure 1. Chord extensions for keyword-based searches

in such a way that routing a message to the destination resembles a distributed binary search. As a result it can efficiently route messages in O (log N) steps for a network consisting of N nodes.

Using the inverted distributed hash table scheme, we can easily extend Chord to support keyword-based queries by maintaining *<keyword, list of values>* information at each DHT node, instead of *<file ID, value>*. Note that because the same keyword can appear in multiple files, the right hand side of the mapping is extended to store a list of values to include the locations of all files containing that keyword. Figure 1 presents an illustration of such an extension.

In this example, there are five files (a.jpg, b.jpg, c.jpg, d.jpg, e.jpg) with corresponding keywords describing the contents of the files. The location information of the files is now distributed using the keyword as the key for consistent hashing. For example, consider b.jpg with a set of keywords {Tree, River, Mountain}. To assign the location information of the file in the extended Chord, each of its keywords is first hashed and assigned to a DHT node. In the example, Tree is assigned to N3, River is assigned to N4, and Mountain is assigned to N1. Since any of these keywords can be used by a query to locate b.jpg, the location information of b.jpg must be stored at all the three nodes, N3, N4, and N1. In this way, we can guarantee that b.jpg's location is returned for any query that searched for Tree, River, or Mountain.

III. KEYWORD FUSION ARCHITECTURE

One of the main challenges in designing a keyword search mechanism for DHT is to address the problem of *common* keywords. Common keywords are those keywords that

frequently appear in the keyword lists of a large number of files. They may provide hints to the file type (e.g. music, picture, mp3, jpg) or are very generic words (e.g. classical, landscape). In this section, we present the main contribution of this paper, namely the *Fusion Dictionary* and *Keyword Fusion* that can effectively handle the problem of common keywords.

A. Preliminaries

Before describing the Keyword Fusion architecture, we first define a few notations. Let *h(k)* denote the hosting DHT node which stores the mapping for keyword *k*, and *K(f)* denote the set of keywords associated with file *f*. Also, let *F(k)* denote the set of files which contains keyword *k*. Throughout this paper we use file *f* and the location of file *f* interchangeably. Using these notations, we can concisely denote the mapping for keyword *k* as <*k, F*(*k*)>. As an example, in Figure 1, *h*(Tree) = *N3*, *F*(Tree) = {a.jpg, b.jpg, d.jpg, e.jpg}, and *K*(c.jpg) = {Boat, River}.

For query processing, we consider only conjunctive queries, i.e. multiple keywords in a query are *AND*-ed. Supporting disjunctive queries (logical *OR*) is achieved by issuing multiple queries. Once a user issues a query, it is routed to the DHT nodes that are responsible for the keywords in the query. At each node, it filters down the query result by intersecting the previous results with its own. For example, in Figure 1, if a user at wants to find files containing *both* Tree and Mountain, he or she can send out a query message to N3 which is responsible for Tree. N3 then sends the intermediate result set {a.jpg, b.jpg, d.jpg, e.jpg} to N1, where the file list of Mountain is stored. By intersecting the intermediate results from N3 with the file list for Mountain, N1 will generate the final result, {a.jpg, b.jpg}. The benefit of this *chained query processing* is documented in [5, 6].

B. Fusion Dictionary & Partial Keywrod List

Consider a query searching for "music *AND* classic *AND* Beethoven". In this query, we observe that music is the most generic keyword, and Beethoven is the most specific keyword. Thus the following inequality holds in terms of cardinality: | *F*(Beethoven) | < | *F*(classic) | < | *F*(music) |. Therefore, when searching for files that contain all three keywords (music, classic, Beethoven), it is advantageous to search for the most specific keyword (Beethoven) first, and then filter the results using the other keywords music and classic. In other words, identifying non-common keywords and processing the query using non-common keywords can optimize the chained query processing. This observation provides an insight into the notion of *Fusion Dictionary*.

Simply put, Fusion Dictionary is a distributed data structure, to which DHT nodes can register common keywords. When a DHT node determines that its storage consumption is excessive based on its local threshold and the other nodes' storage usage learned from control messages, it registers the most common keywords (with the longest value lists in the mapping) into the Fusion Dictionary and removes the entries for the common keyword from its registry. In this respect, Fusion Dictionary can be considered as a collection of common keywords that have been deleted from their hosting nodes. The content of the Fusion Dictionary is replicated and propagated across DHT nodes in order to minimize the lookup overhead. Using the Fusion Dictionary, a query initiating node can transform the user queries to contain only non-common keywords by removing the registered common keywords.

While the Fusion Dictionary can provide an effective guideline to transform the query to contain more specific search terms, this mechanism alone cannot ensure that the query is processed correctly. More precisely, because certain common keywords have been removed from the query, the original semantic of conjunctive query is lost and the returned result will be a superset of the correct result. For instance, suppose that an original query was for "music *AND* classic *AND* Beethoven" and music is a common keyword registered in Fusion Dictionary. Then following the above procedure, the user will send out a query for "classic *AND* Beethoven" and receive a result, in which some of the files do not have music as a keyword.

To address this issue, we introduce a data structure called partial keyword list per file, which contains common keywords (registered in the dictionary) that are associated with the corresponding file. In other words, a partial keyword set for a file *f* is defined as $PK(f) = K(f) \cap FD$. To utilize the partial keyword list, the query is transformed to contain common keywords as meta-information instead of just omitting them. With this modification, now when the query for "music *AND* classic *AND* Beethoven" gets processed at destination nodes for classic and Beethoven, the destination nodes also refer to the meta-information and construct intermediate results to include only the files with keywords classic and Beethoven that also have music in their partial keyword lists.

Managing the Fusion Dictionary and the partial keyword list is a fully decentralized operation in the proposed architecture. Each node maintains a local Fusion Dictionary. The local Fusion Dictionary periodically exchanges heartbeat messages carrying updates with other Fusion Dictionaries of the neighbors. Thus after a well-defined time period *P*, the registration of a keyword *k* to the Fusion Dictionary will be propagated to all DHT nodes[1]. After waiting for time *P*, the hosting node *H*(*k*) removes the file location information from *k's* mapping. A similar decentralized operation is performed to update the partial keyword list. The storage overhead of our mechanism is determined by the keyword set size of each file and the popularity distribution of keywords. A Zipf-like distribution of keyword popularity (See section IV for trace data) indicates that only a small fraction of the keywords are very

[1] The time window *P* can be computed from the update period and the diameter of the DHT network.

common and would be inserted into Fusion Dictionary. Consequently, partial keyword lists would also be small. In addition, the Partial keyword lists can be efficiently encoded since they are small in number and are listed in the Fusion Dictionary.

C. Keyword Fusion

The key insight behind the Fusion Dictionary algorithm is that when a file is associated with multiple keywords *a* and *b*, we can safely remove this file's information from node $h(a)$ as long as the entry for keyword *b* is maintained because the file is still searchable using *b*. Now what happens when $h(b)$ decides that keyword *b* too is generic and must be removed from its hosting DHT node? Such situations are handled by *Keyword Fusion*.

Before we describe Keyword Fusion, we first define a function combine that generates a new keyword by concatenating a set of keywords in the Alphabetic order. Let K denote a set of keywords $\{k_1, k_2, \ldots, k_n\}$. Then combine($K$) generates a new keyword $k' = k_1\&k_2\&\ldots k_n$ where $k_1, k_2, \ldots, k_n$ are enumerated in the Alphabetic order.[2] For example, combine (music, classic) generates a new keyword music&classic. We call them *synthetic* keywords to distinguish them from the original keywords. After a synthetic keyword has been generated a mapping for this new keyword is defined as: $<k_1\&k_2\&\ldots\&k_n, F(k_1)\cap F(k_2)\cap\ldots\cap F(k_n)>$. In other words, the value part of the mapping for the synthetic keyword is a list of the files that contain all $k_1, k_2, \ldots, k_n$ in their keyword lists.

The operation of Keyword Fusion is as follows. Assume Fusion Dictionary contains keywords, $a_1, a_2, \ldots, a_m$. Now suppose a keyword *b* is added into the Fusion Dictionary from its hosting node $h(b)$. New keywords are generated by combining *b* with all the keywords in the Fusion Dictionary and new synthetic keywords are inserted into the P2P network using consistent hashing along with their mappings. More precisely, Keyword Fusion ensures that all the keywords in the Fusion Dictionary that are combined in a pair-wise manner do exist in DHT. For example, if Fusion Dictionary = {*a, b, c*}, Keyword Fusion guarantees that synthetic keywords *a&b, b&c, and a&c* exist in the DHT. Note that synthetic keywords can be further synthesized to generate new keywords if the synthetic keywords are still too common. In this case, they first need to be decomposed and recombined to generate a new keyword in the Alphabetic order.

Now we describe the Keyword Fusion algorithm using an example. Suppose music is in the Fusion Dictionary. Now when keyword classic gets removed from its host h(classic), the hosting node first looks up the partial keyword lists of

```
1  Keyword Fusion (k)
2  {
3    FD ← FD U {k} // fusion dictionary update
4    for all files f in F(k) // iterate through all files containing k
5      for all keywords j in PK(f)
6        L ← decompose (k) U {j}
7        if (all members of 2^L but L are in FD)
8          insert file f using keyword combine (L).
9        end // if
10     end // for all keywords
11   end // for all files
12   remove F(k);
13 }
```

Figure 2. A pseudo code for Keyword Fusion algorithm

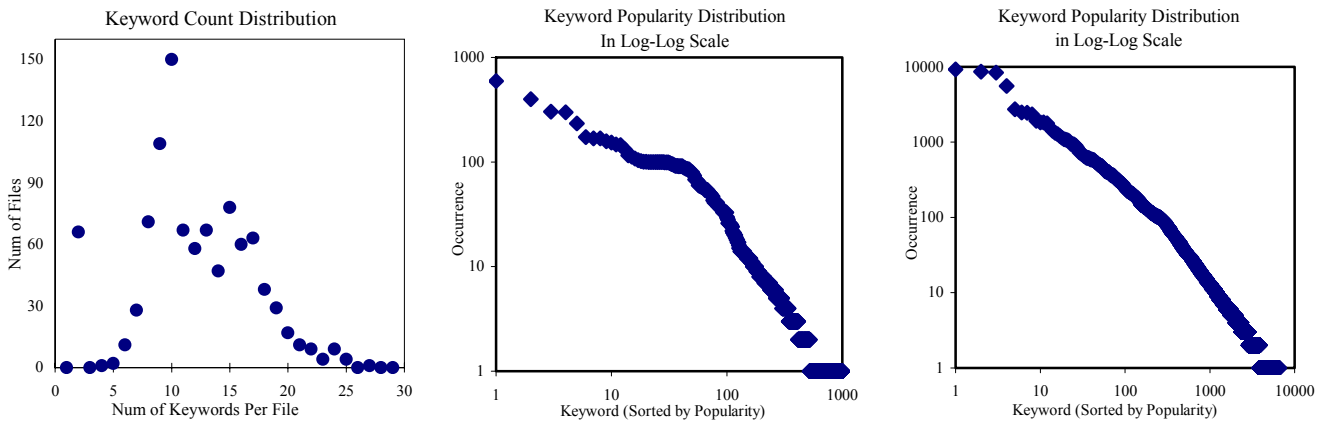
*F(*classic*)* and finds music there. Then before removing the entry for classic, the node create a new keyword music&classic and inserts the mapping <music&classic, F(music) $\cap$ F(classic)> to the inverted DHT network. Once the synthetic keyword and its corresponding mapping information have been successfully inserted, the entry for keyword classic can be removed from its host.

After this Keyword Fusion, suppose a user generates a query for music *AND* classic. The initiating node first looks up its local Fusion Dictionary. Since both music and classic are registered with the Fusion Dictionary, it knows that both keywords have been removed from their original hosts. However, Keyword Fusion guarantees that a new synthetic keyword has been generated for all keywords registered with Fusion Dictionary. Thus, the initiating node modifies the query to music&classic and sends it out to the DHT network. The query will be answered by the node hosting the synthetic keyword music&classic and the result F(music) $\cap$ F(classic) will be returned to the initiating node.

Figure 2 presents a pseudo code for the Keyword Fusion algorithm. Basically, when the mappings for common keyword *k* is removed from $h(k)$, it is first registered with Fusion Dictionary (line 3) and tries to combine with the keywords in the partial keyword lists of *K(f)* (line 4, 5). Line 7 is used to check whether a combination is required. If it is necessary, the new synthetic keywords and their corresponding files are inserted to DHT (line 8) and the entry for the combined files gets removed (line 12). As a result, Keyword Fusion scatters file lists for a common keyword to other nodes. Note that the above algorithm only used the local Fusion Dictionary and the partial keyword list of each file.

To summarize, this section has presented two main ideas of our Keyword Fusion architecture: Keyword Dictionary and Keyword Fusion. The role of Keyword Dictionary is to help peers with excessive storage consumption reduce their loads and help users generate efficient search queries by avoiding common keywords. The role of Keyword Fusion is to ensure that conjunction of deleted keywords can be searched by users by creating synthetic keywords. Both

[2] We maintain the Alphabetic order during keyword fusion to ensure that combine function is commutative, i.e., combine (k_1, k_2) = combine (k_2, k_1). In this way, we ensure that combine (k_1, k_2) and combine (k_2, k_1) are hashed to the same value.



(a) Keyword Count Distribution for date set A (b) Keyword Popularity Distribution for date set A (c) Keyword Popularity Distribution for date set B

Figure 3. Keyword count and popularity distributions in annotated image files

methods, when jointly applied, will reduce the degree of storage imbalance and the possibility of generating large volume query-return traffic.

IV. PERFORMANCE EVALUATION

In this section, we present a simulation-based evaluation of the proposed Keyword Fusion algorithms. In particular, we are interested in the effectiveness of the proposed scheme in terms of traffic reduction, fairness in resource consumption at each peer. For performance evaluation, we have implemented the proposed data structures and algorithms in the Chord simulator [8]. To drive the simulator, we used two data sets: one with 1,000 image files from Corel's image database and the other with 40,000 image files [9, 10].

A. Date set Analysis

Before evaluating our scheme, we first characterize the date set to be used in our simulations. The first date set (we refer to as date set A) includes 1,000 image files that were selected from Corel's image database and manually annotated with relevant keywords by a media lab at the City University of Hong Kong [9]. This image file set contains a total of 1,000 unique keywords and each image is annotated with 12 keywords on the average. The second date set (date set B) is obtained from Digital Library Project in the University of California at Berkley [10] and contains 40,000 images files. More than 38,000 of these files are annotated with four keywords selected from 6,510 unique keywords.

Figure 3 presents the characteristics of the two date sets. The graph (a) shows the keyword count distribution for date set A, i.e. number of files vs. number of keywords attached to each file. From the figure, we observe that the distribution follows a bell shape in general with average of 12 or 13 keywords per file. We also note that as an exception, more than 60 files have only two keywords. On the other hands, in date set B we observed that almost all files have about four keywords.

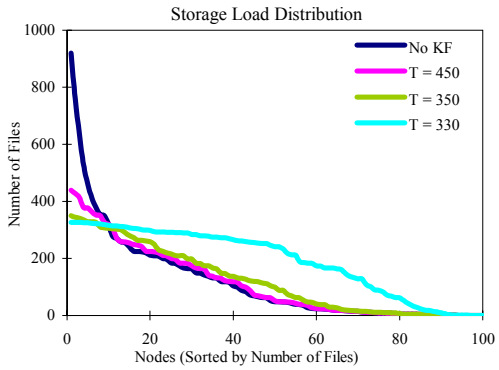
Figure 3 (b) and (c) present the frequency of each keyword occurring in each file's annotation for the two date sets. As we expected, these two graphs show that popularity of keywords used for annotation roughly follows a Zipf-like distribution. In date set A, the highest rank keyword is used to annotate more than 300 files among total 1,000 files. We observe that the top 5% most frequent keywords (i.e. 50 words) appear 6,608 times, which is about a half the total annotations. In date set B, top 5% most frequent keywords appear 124,534 times out of the total 161,051 keyword occurrences. Our analysis shows that hosting nodes for these 5% most frequent keywords will consume excessive amount of storage and extra processing cycles to handle frequent updates in the keyword information due to file addition and deletion.

B. Impact of Keyword Fusion

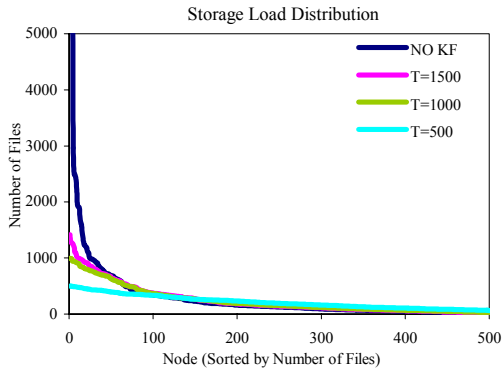
In our architecture, sending search queries for synthetic keywords instead of searching the original keywords should reduce network bandwidth consumption. This results from the fact that synthetic keywords are typically mapped to a much smaller file list than the original ones. In this subsection, we briefly illustrate the impact of this Keyword Fusion in terms of reduction in file information. Table 1 presents several sample keyword pairs and the size of resulting file list when Keyword Fusion has been applied to

Keyword 1 (K1)	#Files	Keyword 2 (K2)	#Files	K1+K2	K1&K2	Reduction
PLANT	116	FLOWER	101	217	101	53%
PLANT	116	LEAVES	147	263	72	62%
SNOW MOUNTAIN	91	SKY	153	244	67	58%
SUNSHINE	85	PEOPLE	145	230	46	65%
FLOWER	91	FRAGRANCE	99	190	91	50%
BEACH	101	SKY	153	254	49	67%

Table 1. Reduction of search result traffic using Keyword Fusion



(a) Date set A with 100 nodes



(b) Date set B with 1,000 nodes

Figure 4. File list redistribution using Keyword Fusion

date set A. From the table, we observe that Keyword Fusion can reduce the search result traffic by up to 67% compared to the regular chained query processing. Note that our example is a conservative case of combining only two common keywords that are correlated. In practice, synthesizing a new keyword from multiple keywords that are weakly correlated will further improve the reduction factor.

C. Balancing Storage Demand

In this subsection we present the effectiveness of Keyword Fusion in terms of reduction of resource consumption at DHT nodes that host common keywords.

Figure 4 presents the amount of file information that has been assigned to each participating peers using the vanilla extension of Chord, and using Keyword Fusion with various threshold values. Figure 4 (a) presents the case of date set A under extended Chord consisting of 100 nodes while Figure 4 (b) is the case that date set B is inserted into 1000 nodes. Note that the nodes are sorted according to their storage consumption and only a portion of the nodes are shown in the graph as the curves are slowly and constantly tapering down for the rest of the nodes in Figure 4 (b). In the figures, "No KF" represents the case when Keyword Fusion has not been used. In this case, we observe that the load is highly skewed and concentrated on a few nodes; in Figure 4 (a), top 5% highly-loaded nodes store 3,307 file entries (660 files per node) and top 10% highly-loaded nodes store 5,121 file entries (512 file entries per node) where the average is only 120 file entries per node. In Figure 4 (b), the case is even worse; top 5% highly-loaded nodes stores 87,884 file entries, which are about 55% of total file entries.

In each graph, the other three curves correspond to the case of Keyword Fusion with three different thresholds. For example, in Figure 4 (a), for three cases where the threshold T is set to 450, 350, and 330, respectively, the number of files hosted by overloaded nodes successfully reduces below the threshold. More specifically, when T is set to 350, the top 5% nodes now store only 1,698 files, about 50% reduction compared to the "No KF" case. Notice that in the case of T = 330, the distribution is more leveled, but the average number of files per node increases. This is the case when the system is overcorrected by setting threshold value too low. Similarly, Figure 4 (b) presents the results with date set B having threshold set to 1500, 1000 and 500. This graph shows a similar result as with date set A. However, in this case a low threshold will not generate a large number of new instances.

Overall the simulation results demonstrate that Keyword Fusion can effectively reduce the excessive storage consumption at overloaded peers and redistribute the storage demand to the file sharing system, without significantly increasing the storage consumption on the other peers.

V. RELATED WORK

The P2P environment recently has become one of the most popular information sharing architectures. Although search technologies on the Web have been studied extensively, their centralized indexing scheme is not suitable for highly dynamic P2P systems. Providing searching capability is quite different for two different P2P models: unstructured and structured. Unstructured P2P file sharing systems, such as Gnutella [11], uses flooding as its links are naturally constructed by user selected neighbors. While it can easily provide keyword or attribute-based search, its efficiency and coverage are limited. On the other hand, structured P2P systems, such as Tapestry [4], Pastry [2], Chord [1], and CAN [3] guarantee locating existing files, regardless of their physical and logical locations. However, they are all designed to locate a file with its unique index key and are not capable of searching with keywords.

A few studies [5, 6] extended the DHT scheme to support keyword search using the inverted DHT. With such extension, keyword-based search can return the list of all relevant files, but its query traffic can cause a significant burden on the network. Reynolds et al. [5] addressed this

problem by processing queries cooperatively at destination nodes and using bloom filters to compress intermediate file lists. They also cache the results of previous queries to further reduce network traffic and response time. Panache [6] proposed truncating intermediate results using file popularity information. However, these approaches do not address the problem introduced by common keywords.

Gnawali [12] proposed Keyword-Set Search (KSS), which groups multiple keywords into a keyword-set and uses it as a hash index. This is similar to our work in that keywords are combined. However, KSS can create excessive redundant file lists as all possible keyword combinations are generated for each file insertion. In contrast, our Keyword Fusion combines keywords adaptively depending on their changing popularity.

VI. CONCLUSION

In this paper, we have proposed a set of mechanisms to provide a scalable keyword-based file search in a DHT-based P2P system. Our proposed solution, called Fusion Dictionary and Keyword Fusion, balances unfair storage consumptions at peers, transforms users' queries to contain focused search terms. To evaluate the performance, we have implemented the Keyword Fusion algorithm by extending the Chord Simulator and used two types of data sets. The results show that Keyword Fusion can reduce search traffic by up to 67% even in a modest scenario of combining two relatively common keywords. We have also shown that Keyword Fusion can effectively alleviate overloaded peers and distribute the file storage load across the entire DHT network. For example, Keyword fusion reduces the storage consumption of the top 5% most loaded nodes by 50% without significantly increasing the storage consumption level in the other peers.

We are currently conducting more extensive experiments using various date sets, query patterns, and keyword correlations. We are also developing a distributed algorithm to dynamically adapt the threshold value for Key Fusion.

REFERENCES

[1] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications," *Proc. SIGCOMM'01*, 2001.

[2] A. Rowstron and a. P. Druschel, "Pastry: Scalable, distributed object address and routing for large-scale peer-to-peer systems," *Proc. IFIP/ACM Int'l Conf. Distributed Systems Platforms,* 2001.

[3] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A Scalable Content-Addressable Network," P*roc.* A*CM SIGCOMM 2001,* 2001.

[4] B. Y. Zhao, J. Kubiatowicz, and A. D. Joseph, "Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing," *IEEE Journal on Selected Areas in Communications (c) 2003 IEEE,* 2001.

[5] P. Reynolds and A. Vahdat, "Efficient Peer-to-Peer Keyword Searching," *Proc. Middleware. 2003,* 2003.

[6] T. Lu, S. Sinha, and A. Sudan, "Panache: A Scalable Distributed Index for Keyword Search," 2003.

[7] G. Zipf, Selective Studies and the Principle of relative Frequency in Language, C. Harvard University Press, MA, 1932, Ed., 1932.

[8] Chord, "http://www.pdos.lcs.mit.edu/chord/."

[9] Benjiman, "http://abacus.ee.cityu.edu.hk/~benjiman/corel_1/."

[10] Digital-Library-Project, "http://elib.cs.berkeley.edu/photos/corel/."

[11] Gnutella, "http://genutella.wego.com."

[12] O. D. Gnawali, "A Keyword-Set Search System for Peer-to-Peer Networks," in *MIT.* Massachusetts, 2002.