Peer-to-Peer Keyword Search Using Keyword Relationship

Kiyohide Nakauchi Yuichi Ishikawa Hiroyuki Morikawa Tomonori Aoyama
School of Engineering, The University of Tokyo
{nakauchi, ishikawa, mori, aoyama}@mlab.t.u-tokyo.ac.jp

Abstract

Decentralized and unstructured peer-to-peer (P2P) networks such as Gnutella are attractive for Internet-scale information retrieval and search systems because they require neither any centralized directory nor any centralized management of overlay network topology and data placement. However, due to this decentralized architecture, current P2P keyword search systems lack useful global knowledge such as popularity of data items and relationships between keywords and data items. As a result, current P2P keyword search systems supports only naive text-match search and can find only data items with a keyword (or meta-data) exactly indicated in a query.

In this paper, we show an efficient P2P search system which increases possibility of discovering desired data items. The key mechanism is query expansion, where a received query is expanded based on keyword relationships managed in a distributed fashion by participating nodes. Keyword relationships are improved through search and retrieval processes and each relashionship is shared among nodes holding similar data items. We also present implementation of our P2P search system.

1. Introduction

As the number of contents on the Web grows into the billions, the larger and larger portions of those contents are not accessible through centralized search engines [1, 2]. Recent Estimates [3] in March 2000 place the size of deep Web nearly 500 times the size of publicly indexable Web. The deep Web continues to grow at a remarkable rate, as the more existing databases are online, over which Web-accessible search facilities are provided. This fact shows the necessity of search infrastructure scaling at a comparable rate.

Peer-to-peer (P2P) systems are now one of the most prevalent Internet distributed applications due to their scalability, fault-tolerance, and self-organizing nature. This trend was triggered in 1999 by Napster [4], a centralized architecture, where a central directory server offers an index to locate data items. However, legal issues of Napster inherent in its centralized architecture shifted the interests of the research community and Internet users to a decentralized architecture, where the index, query processing, and content transfer are fully distributed among nodes.

The primarily focused problems of decentralized architectures are scalability and partial-match lookup capability. One class of the decentralized architectures is an unstructured P2P system such as Gnutella [5], where the overlay topology is formed in accordance with some loose rules [6]. Data items are indexed locally and the query can be resolved only by the nodes that holds them. Due to this loose structure, queries are typically flooded. Thus unstructured P2P systems are flexible enough to support partial-match lookup or other text-based search mechanisms, and are very robust against nodes' failure and removal. Though the systems achieve poor scalability due to query flooding, recently some efforts are made to improve scalability. For example, FastTrack [7] organizes subscribing nodes into loosely hierarchical structure where some nodes are selected as supernodes and cache the index. Multiple random walks [8] and associative overlays [9] reduce the number of forwarded queries by limiting searches to a fraction of the population.

The other class of the decentralized architectures is a structured P2P system commonly referred to as Distributed Hash Tables (DHTs) [10, 11, 12, 13], where the overlay topology is tightly controlled. The nodes that are required to store or index data items are precisely determined based on some hashing algorithms. This tightly controlled structure enables to forward queries deterministically, and achieves very effective content location. While structured P2P systems are highly scalable for locating data items with unique item identifier, inherently they cannot efficiently provide partial-match lookup capability. Some works [14, 15] try to achieve partial-match lookup on DHTs. In addition, it is widely recognized that due to their tight control, DHTs

Proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'03)
0--7695-1919-9/03 $17.00 © 2003 IEEE

incur much larger overhead to re-organize the overlay network than unstructured P2P systems when nodes fail or leave the network.

As described above, current P2P systems are actively improved by the research community to achieve both scalability and partial-match lookup capability simultaneously. However, search algorithms of current P2P systems still seem not to be efficient enough when compared to the sophisticated state-of-the-art IR algorithms. This is because current P2P keyword search systems lack useful global knowledge such as popularity of data items and relationships between keywords and data items, which would be difficult to obtain or compute in decentralized architectures. That is, while current P2P systems support naive text-match search, they cannot support *semantic search*. As a result, the systems can only find data items which are given a keyword (or meta data) exactly indicated in a query.

The most familiar mechanism enabling semantic search is *query expansion* [16, 17], which has been investigated as an IR technique for several decades. Query expansion means adding relevant terms to the original query. The purpose of query expansion is to cope with the mismatch between the term used by searcher and that expected by writers of the documents. This mismatch may be due to synonymy, where different terms have the same meaning, or granularity, where terms are used at different levels of detail. We believe that in P2P search systems, query expansion could significantly improve the possibility to locate desired data items because most of the contents current popular P2P search systems are dealing with are multimedia contents, which have only several keywords in general.

Our goal is to build an efficient decentralized P2P search system that supports semantic search by query expansion, while retaining desirable properties of prevailing unstructured P2P systems such as simplicity and robustness. In our system, not only well-defined items, which are given keywords easy to imagine for searchers, but also poorly-defined items, which are given keywords generally or unexpectedly difficult to image for searchers, can be found efficiently.

In traditional IR systems, query is expanded generally based on some kind of database such as a thesaurus, which keeps relationships between terms extracted from the statistics computed in advance using samples representative of enough amounts of documents. In P2P keyword systems, however, because of the decentralized nature, it is not desirable that any centralized node calculates the statistics to obtain keyword relationships or keeps the thesaurus. We believe, even if the complete thesaurus exits, the thesaurus, which may be huge enough for a node to store, should be divided and each portion should be kept among distributed nodes in order to retain the general advantages of P2P systems such as robustness and load balancing. We also believe that keyword relationships, though they may change at relatively slow rate, should be computed distributedly for the same reason. Thus the key challenge of our P2P search systems is to construct the thesaurus in a fully distributed manner.

In this paper, we describe the basic mechanism of P2P keyword search using keyword relationships and the distributed mechanisms of updating a thesaurus. In our system, we name the thesaurus Keyword Relation DataBase or KRDB. Each node holds a KRDB, and the KRDB keeps keyword relationships which are relevant only for the data items the node stores. The accuracy of KRDBs significantly affects the search performance. We propose two KRDB update mechanisms; evaluation feedback and KRDB synchronization.

In general, search with query expansion leads to increase, or in the worst case, implosion of search results. To cope with this problem, in addition to KRDB improvement, we introduce results ranking, where the data items with more relevant keywords with a query are ranked higher. Fortunately, we can leverage KRDBs for this purpose by introducing the concept of "strength" for each keyword relationship. We believe the ranking algorithm presented in this paper would also benefit traditional P2P search systems.

Several features that differentiate our P2P keyword search system from others are as follows.

- It is controlled in a fully distributed fashion, without any single point of failure and any centralized management and placement of the thesaurus.
- It supports semantic search in addition to naive text-match search.
- It enables results ranking.

The reminder of this paper is organized as follows. In Section 2, we describe the overview and basic mechanisms of P2P keyword search with query expansion based on keyword relationships. Section 3 describes the distributed update mechanisms of KRDBs to enhance search performance. We present implementation of our P2P search system in Section 4, and conclude in Section 5.

2. P2P Search with Query Expansion

The fundamental idea in our P2P keyword search mechanism is query expansion using KRDB managed and updated in a fully distributed manner. In this section, we describe

how to create a KRDB and then show basic P2P search mechanism. We also present ranking algorithm.

2.1. KRDB

A KRDB is a thesaurus which keeps some information about keywords relevant only to the data items stored locally in a node. That means each node may have a different and minimum KRDB. This distributed KRDB management can clearly retain the desirable properties of P2P systems.

The most important information on keywords in a KRDB is keyword relationship (KR) of each pair of keywords and its strength (KRStr). In this paper, $\mathrm{KR}(\mathtt{k}_i, \mathtt{k}_j)$ denotes a keyword relationship from keyword $\mathtt{k}_i$ to keyword $\mathtt{k}_j$ ($0 \leq \mathrm{KR}(\mathtt{k}_i, \mathtt{k}_j) \leq N$, where N denotes the maximum number of keywords in a KRDB). In other words, $\mathrm{KR}(\mathtt{k}_i, \mathtt{k}_j)$ is defined as follows; when keyword $\mathtt{k}_i$ is given, keyword $\mathtt{k}_j$ is referred to as a relevant term to keyword $\mathtt{k}_i$. Note that $\mathrm{KR}(\mathtt{k}_i, \mathtt{k}_j)$ and $\mathrm{KR}(\mathtt{k}_j, \mathtt{k}_i)$ should be distinguished from each other. The other variables in a KRDB are shown in Section 3.

Before we explain the algorithm of extracting KRs, we describe our insight for them. Our insight is such that keywords given to a data item are relevant. In the current P2P search systems, we assume data items are mainly multimedia contents such as audio and video, which have generally much fewer keywords than documents because they have much less textual contents. Therefore, we consider these keywords represent the characteristics of the data items more precisely and keyword relationships would be helpful for finding a limited number of other meaningful keywords. In addition, the extracted KRs are just initial state, and gradually improved through KRDB update mechanisms shown in Section 3.

Figure 1 shows how to create KRs between keywords. There are two processes to create KRs. First, when a node joins the P2P network, the node firstly extract all the keywords for each local data item. For example, the node takes out four keywords $\mathtt{A}, \mathtt{B}, \mathtt{C}, \mathtt{D}$ from data item 1. We consider these keywords have relationships between each other. Following the definition of $\mathrm{KR}(\mathtt{k}_i, \mathtt{k}_j)$, twelve KRs are created from data item 1. In the same way, twelve and six KRs are created from data item 2 and 3 respectively. Each $\mathrm{KR}(\mathtt{k}_i, \mathtt{k}_j)$ keeps $\mathtt{KRStr}(\mathtt{k}_i, \mathtt{k}_j)$, which denotes the strength of $\mathrm{KR}(\mathtt{k}_i, \mathtt{k}_j)$ ($0 \leq \mathtt{KRStr}(\mathtt{k}_i, \mathtt{k}_j) \leq 1$). Larger $\mathtt{KRStr}(\mathtt{k}_i, \mathtt{k}_j)$ means $\mathrm{KR}(\mathtt{k}_i, \mathtt{k}_j)$ is stronger, and $\mathtt{KRStr}(\mathtt{k}_i, \mathtt{k}_i)$ is set to 1. KRStr is updated to reflect more accurate KR based on both evaluation feedback and KRDB synchronization described in Section 3, and is used for results ranking. The initial value of KRStr is *KRStrInit* (0.5 in our system).

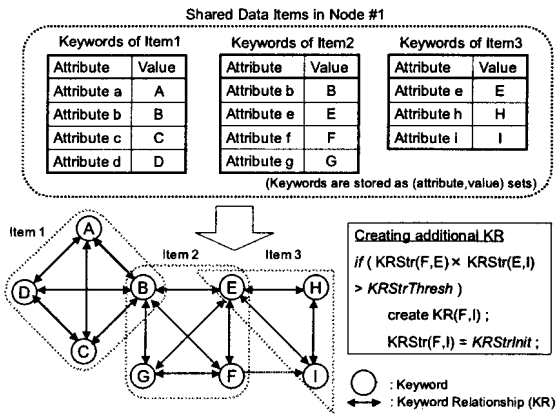

Figure 1. Creating Keyword Relationships from Local Data Items

Then, additional KRs are created, if two KRs share a common keyword. For example, as shown in Figure 1, if the value of $\mathtt{KRStr}(\mathtt{F}, \mathtt{E}) \times \mathtt{KRStr}(\mathtt{E}, \mathtt{I})$ is larger than the pre-defined threshold *KRStrThresh*, $\mathrm{KR}(\mathtt{F}, \mathtt{I})$ is newly created. However, less useful KRs are removed from KRDBs to prevent the waste of storages and computation power (see 3.2).

2.2. Basic P2P Search Mechanism

In our P2P keyword search system, in the same way as traditional unstructured P2P systems, overlay topology is organized in accordance with some loose rules [6]. A query includes several keywords and is flooded to be resolved. This unstructured architecture enables the system to retain desirable properties such as simplicity and robustness against node's failure or removal.

The key mechanism of our P2P keyword search system, which differentiates our system from traditional unstructured P2P systems, is query expansion at nodes that receive the query. Figure 2 shows the basic search mechanism using query expansion.

An alternative approach would be query expansion only at a searcher. A query is not expanded at nodes that receive the query. This approach, however, would work well only if a searcher desires data items with the same keywords as those in its local KRDB. Otherwise, the query is not expanded anywhere, that leads to the same search results as those of traditional non-semantic P2P search.

When a node join the P2P network, the node first con-

structs a KRDB in the way described in Section 2.1. The search process is as follows.

1. A searcher sends a query which indicates several keywords. This query is flooded (forwarded peer-to-peer) with certain TTL. Note that the forwarded query is identical with the received query (query expansion affects only local search), because consecutive query expansion at different nodes for leads to query explosion with less relevant keywords, so that the possibility increases to find less desired data items.

2. A node that receives the query performs query expansion using a local KRDB. Specifically, the original query is expanded to include several additional keywords to which there is a KR from keywords in the original query. For example, at Node #1 in Figure 2, there exists KR(`red`, `apple`), so keyword `red` is expanded to two keywords `red` + `apple`. In the same way, keyword `fruits` is expanded to two keywords `fruits` + `apple`. As a result, the original query is expanded to the query `red` + `fruits` + `apple`.

3. Node #1 searches local data items using the expanded query. If there exists any data item which has one or more keywords of the expanded query, Node #1 replies to the searcher with search results (a list of satisfied data items) using a QueryHit message.

4. The searcher gathers search results and ranks all the located data items. The ranking algorithm is shown in 2.3. Then the searcher selects one or more desirable data items, and search itself is completed.

5. At the same time, the searcher feedbacks the evaluation to all the nodes which returned search results obtained using KR(`red`, `apple`) or KR(`fruits`, `apple`) (Node #1 and #2 in Figure 2) for the purpose of updating KRDBs in those nodes. Evaluation results indicate the KRs which are used to locate the selected data items. The details of evaluation feedback are described in 3.1.

2.3. Results Ranking

The search results are ranked using `KRStr` between a keyword in an original query and that given to the located data item. The basic idea of results ranking is that when an original query includes keyword `k`, the rank of the data item gets higher as $\texttt{KRStr}(\texttt{k}, \texttt{l})$ is larger, where `l` denotes a keyword given to the data item. If several keywords are

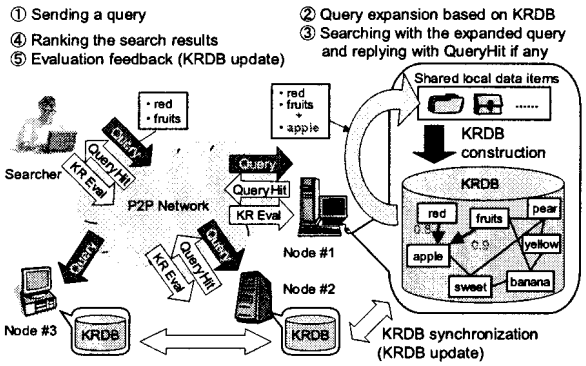

Figure 2. Basic P2P Search Mechanism

included in a query, or are given to data items, the above ranking algorithm is changed as follows; when an original query includes keywords $\texttt{k}_\texttt{i}(\texttt{i} = 1, 2, ...)$, the rank of the located data item gets higher as simply $\sum_{i,j} \texttt{KRStr}(\texttt{k}_\texttt{i}, \texttt{l}_{\texttt{i}'})$ is larger, where $\texttt{l}_{\texttt{i}'}$ $(\texttt{i}' = 1, 2, ...)$ denote keywords given to the data item. Note that other sophisticated ranking algorithms using `KRStr` could be applied. We believe these ranking algorithms would also benefit traditional P2P search systems only for the purpose of results ranking by introducing KRDB-like databases unnoticeable for users into the search systems. Such application is beyond the scope of this paper.

3. Distributed KRDB Updates

In our P2P keyword search system , the accuracy of KRDBs significantly affects the search performance. Therefore KRDBs are required to be updated and kept as accurate as possible. In this paper, we show two KRBD update mechanisms; evaluation feedback and KRDB synchronization. Evaluation feedback aims to improve KRDBs through a search process. In this mechanism, subjective evaluations of searchers are directly reflected to KRDBs, and potential statistical effects could be expected. On the other hand, KRDB synchronization aims to improve KRDBs which include inaccurate information due to its special environments where, for example, the node has just joined the P2P network, or the node has just exposed a lot of new data items. These two update mechanisms are complementary to each other and are essential for keeping KRDBs accurate.

3.1. Evaluation Feedback

Evaluation feedback updates $\texttt{KRStr}(\texttt{k}_\texttt{i}, \texttt{k}_\texttt{j})$ in the nodes which replied to the searcher with resulted data items found

Table 1. Variables in a KRDB

$\mathtt{PKFlag}(k_i)$	If k_i is PK, $\mathtt{PKFlag}(k_i) = 1$. Else, $\mathtt{PKFlag}(k_i) = 0$.
$\mathtt{KRStr}(k_i, k_j)$	Strength of $\mathrm{KR}(k_i, k_j)$. $(0 \leq \mathtt{KRStr}(k_i, k_j) \leq 1)$
$\mathtt{UsedCnt}(k_i, k_j)$	Count that $\mathrm{KR}(k_i, k_j)$ is used for search.
$\mathtt{HelpfulCnt}(k_i, k_j)$	Count that $\mathrm{KR}(k_i, k_j)$ is evaluated to be helpful.

using $\mathrm{KR}(k_i, k_j)$. The basic idea of evaluation feedback is as follows; when a searcher initiates a query with keyword k_i and then selects a data item with keyword k_j from the resulted item list (the item was located by query expansion using $\mathrm{KR}(k_i, k_j)$), $\mathrm{KR}(k_i, k_j)$ is regarded as helpful and $\mathtt{KRStr}(k_i, k_j)$ is increased. Otherwise, $\mathtt{KRStr}(k_i, k_j)$ is decreased.

In Table 1, we show the variables kept in a KRDB. $\mathtt{UsedCnt}(k_i, k_j)$ is incremented when both of the following conditions are satisfied; 1) the original query including keyword k_i is expanded using $\mathtt{KRStr}(k_i, k_j)$ and 2) a data item with keyword k_j is located. On the other hand, $\mathtt{HelpfulCnt}(k_i, k_j)$ is incremented only when additionally the following third condition is satisfied; 3) a data item with keyword k_j is selected by a searcher. We define $\mathtt{KRStr}(k_i, k_j)$ using these two variables;

$$\mathtt{KRStr}(k_i, k_j) = \frac{\mathtt{HelpfulCnt}(k_i, k_j)}{\mathtt{UsedCnt}(k_i, k_j)}$$

This means that $\mathtt{KRStr}(k_i, k_j)$ increases as more searchers regards $\mathrm{KR}(k_i, k_j)$ as useful ($\mathtt{HelpfulCnt}(k_i, k_j)$ increases).

Figure 3 show the process of evaluation feedback. In this figure, a searcher initiates a query including keywords A and B.

1. Each node receives a query with keyword k_i from a searcher. At this time, a node which keeps $\mathrm{KR}(k_i, *)$ ($*$ denotes an arbitrary keyword) increments $\mathtt{UsedCnt}(k_i, *)$. In Figure 3, $\mathtt{UsedCnt}(\mathtt{A}, \mathtt{W})$ and $\mathtt{UsedCnt}(\mathtt{B}, \mathtt{X})$ are incremented at Node #1 and #3, and at Node #1 and #2 respectively. More specifically, for example, Node #1 initially has $\mathtt{KRStr}(\mathtt{A}, \mathtt{W})$ = 3/6 (= 0.5), that means $\mathtt{UsedCnt}(\mathtt{A}, \mathtt{W})$ = 6 and $\mathtt{HelpfulCnt}(\mathtt{A}, \mathtt{W})$ = 3. Then, at Node #1 $\mathtt{UsedCnt}(\mathtt{A}, \mathtt{W})$ and $\mathtt{UsedCnt}(\mathtt{B}, \mathtt{X})$ are incremented from 6 to 7 and from 7 to 8 respectively.

2. Each node notifies the searcher of search results with $\mathrm{KR}(k_i, *)$ used for locating the data item. In Figure 3, for example, Node #1 notifies the searcher of Item 1 with $\mathrm{KR}(\mathtt{A}, \mathtt{W})$ and Item 2 with $\mathrm{KR}(\mathtt{B}, \mathtt{X})$.

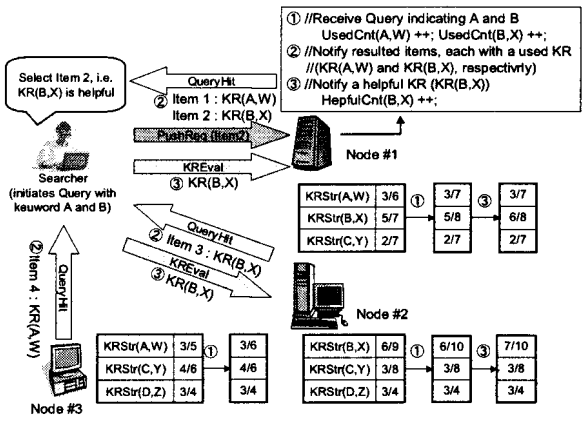

Figure 3. KRDB Update by Evaluation Feedback

3. When a data item without keyword k_i (with keyword k_j) is selected by the searcher among search results, the searcher notifies all the responsive nodes by unicast who keep the same KR ($\mathrm{KR}(k_i, k_j)$) used to locate the data item. At this time, the node that receive this evaluation feedback increment $\mathtt{HelpfulCnt}(k_i, k_j)$ of the KRs specified by the feedback.

   In Figure 3, the searcher selects data item 2 among search results. Data item 2 is found using $\mathrm{KR}(\mathtt{B}, \mathtt{X})$ at Node #1, so the searcher refers to $\mathrm{KR}(\mathtt{B}, \mathtt{X})$ as a helpful relationship. Then the searcher sends evaluation feedback (using a KREval message) to Node #1 and #2, who return the data item found using $\mathrm{KR}(\mathtt{B}, \mathtt{X})$. When Node #1 and #2 receive evaluation feedback, they increment $\mathtt{HelpfulCnt}(\mathtt{B}, \mathtt{X})$ from 5 to 6, and from 6 to 7 respectively. Note that $\mathtt{HelpfulCnt}(\mathtt{A}, \mathtt{W})$ retains the same value because $\mathrm{KR}(\mathtt{A}, \mathtt{W})$ is useless for the searcher.

   Thus, strength of the more helpful relationship ($\mathtt{KRStr}(\mathtt{B}, \mathtt{X})$) increases , while that of the less helpful relationship ($\mathtt{KRStr}(\mathtt{A}, \mathtt{W})$) decreases. $\mathtt{KRStr}(\mathtt{C}, \mathtt{Y})$ and $\mathtt{KRStr}(\mathtt{D}, \mathtt{Z})$ remain as it is, because keyword C and D are not included in the query.

Through this evaluation feedback process of all the searchers, each KRStr in KRDBs is gradually and statistically refined.

3.2. KRDB Synchronization

The KRDB update mechanism by evaluation feedback is efficient for improving the accuracy of KRStr. However, evaluation feedback has two drawbacks. First, evaluation feedback can only evaluate the existing KRs, which are extracted only from local data items. Second, while evaluation feedback can basically improve the accuracy of KRDBs, it would take a long time to make the value of KRStr statistically meaningful. For this reason, the nodes with short lifetime or the nodes who have just begun to expose new data items cannot provide accurate KRDBs soon.

To overcome these drawbacks, we propose another KRDB update mechanism, KRDB synchronization, where familiar KRs and statistically more accurate value of KRStr are shared among nodes. The basic idea of KRDB synchronization is as follows; 1) relevant KRs to a node are added to its KRDB, and 2) when common KRs are kept at some nodes, the value of KRStr at each node are updated to the most accurate one.

3.2.1 With Which Nodes Are KRDBs Synchronized ?

In order to efficiently improve KRDBs, it is required for nodes to synchronize their KRDBs with those in adequate nodes. Considering straightforwardly, adequate nodes might be nodes which holds as many identical or similar data items as possible, because consequently they are likely to keep identical KRs. However, it would be laborious to check all data items in other nodes. In addition, it would be difficult to judge remotely whether two data items are identical or not when their names are different.

Then, we refer to keywords as abstraction of data items, and we regard nodes with more common keywords in their KRDBs as adequate nodes to synchronize with. Here, we call the keywords which can be extracted only from local data items Primary Keywords or PKs in order to distinguish them from additional ones (Secondary Keywords, or SKs) that are added by KRDB synchronization. Based in this discussion, we define an adequate node to synchronize with as the node where the number of the common PKs is larger than the threshold *SynchThresh*.

In this paper, the way to discover the adequate nodes is simply broadcasting. These broadcasted messages could be merged with query messages. Therefore we consider this methodology does not degrade scalability. Note that more sophisticated and scalable discovery algorithms such as multiple random walks [8] would be applicable. We leave this issue as a future work.

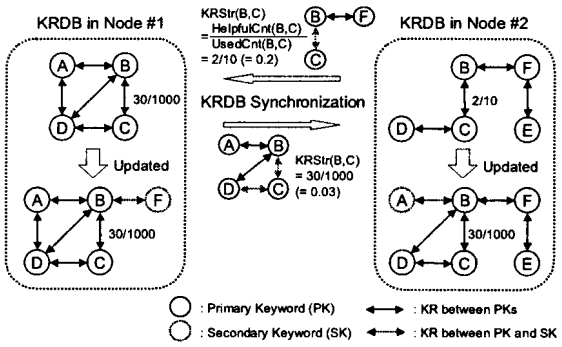

Figure 4. KRDB Synchronization

3.2.2 Synchronization Mechanisms

Subscribing nodes periodically synchronize their KRDBs with adequate nodes. Figure 4 shows an overview of the KRDB synchronization mechanism. At first, a node discovers several adequate nodes for KRDB synchronization in the way described above. KRDB Synchronization consists of two mechanisms. The first mechanism is KR addition, where a KR between two PKs or a KR between a PK and a SK kept in one node is added to the KRDB in the other node only when one of the two PKs or the PK is common for those two nodes respectively. In order to avoid KR explosion, a KR between two SKs in one node is not added to the other node.

For example, in Figure 4, the KRDB in Node #1 is updated by adding two KRs concerning PK B, KR(B, F) and KR(F, B), which are extracted from the KRDB in Node #2, because there exists a common PK B. In the same way the KRDB in Node #2 is updated by adding four KRs concerning PK B (i.e. KR(B, D), KR(D, B), KR(B, A), and KR(A, B)).

KR addition, however, would still have a risk of KR explosion due to adding useless or meaningless KRs with common PKs. To further alleviate KR explosion, we remove such KRs. Specifically, we remove both useless KRs with KRStr smaller than the threshold *KRStrMin* and meaningless KRs with UsedCnt smaller than the threshold *UsedCntMin*

The second is KRStr modification , where KRStr in one node is updated to that in the other node, which would be statistically more accurate, when KRs between two PKs are common for those two nodes. Here, we consider that accuracy of $\mathtt{KRStr}(\mathtt{k}_i, \mathtt{k}_j) (= \frac{\mathtt{HelpfulCnt}(\mathtt{k}_i, \mathtt{k}_j)}{\mathtt{UsedCnt}(\mathtt{k}_i, \mathtt{k}_j)})$ statistically increases as $\mathtt{UsedCnt}(\mathtt{k}_i, \mathtt{k}_j)$ increases. For example, in Figure 4, KRStr(B, C) in Node #2 is updated from 2/10 (= 0.2)

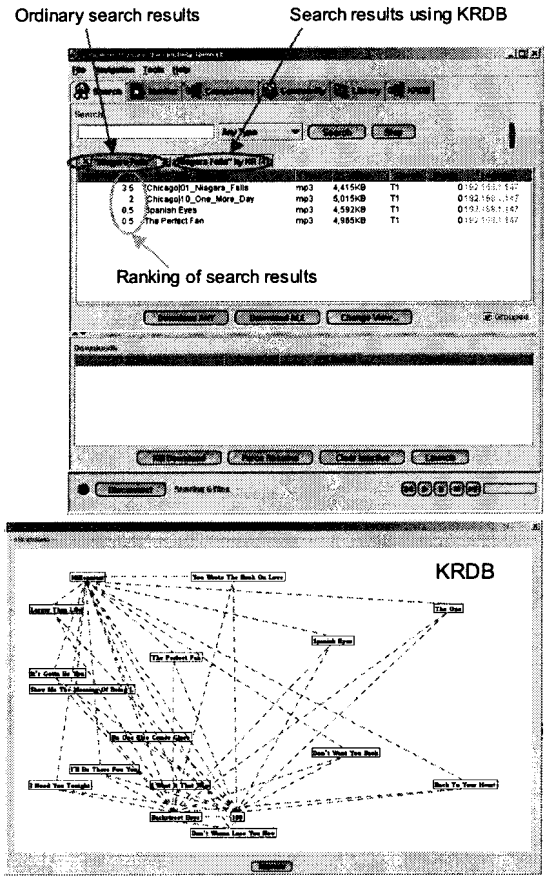

Figure 5. Implementation on LimeWire

to 30/1000 (= 0.03) because `UsedCnt(B, C)` in Node #1 (= 1000) is larger than that in Node #2 (= 10).

4. Implementation

We are now developing a prototype of P2P search system with query expansion. We have implemented the search algorithms described in Section 2 and 3 on LimeWire [18], which is one of the most famous open-source P2P keyword search applications and written in Java.

Figure 5 is a screenshot of the prototype (LimeWire with query expansion). The upper window shows the ranked search results. In order to make clear the difference between search results of normal search and that with query expansion, and to help performance evaluation and further development, the prototype executes normal search and search with query expansion simultaneously. Note that because the query routing algorithm is common for these two search mechanism, no more traffic is generated. We make it possible to switch the display for search results of these two mechanisms. On the other hand, the lower window shows the visualized KRDB. The dotted line means the KRs and quadrangles indicate keywords.

We are now running the prototype at several desktop PCs in the local area network, and further plan to run it at several locations globally in the Internet. We are about to extensively evaluate the performance by system and traffic monitoring, including overhead of computation and traffic for both query expansion and KRDB updates. We also try to make clear the advantages of query expansion quantitatively, though it would be difficult without extensive and continuous uses. Performance evaluation of such P2P search systems would be relatively easy through computer simulations.

5. Conclusion

In this paper, we described the basic concept of an efficient decentralized P2P search system that supports semantic search by query expansion, while retaining desirable properties of prevailing unstructured P2P systems such as simplicity and robustness. In this P2P search, queries are expanded based on KRDBs to improve the possibility to find a desired data item. We proposed a results ranking mechanism to cope with consequent results implosion. In order to improve KRDBs and to enhance search performance, we proposed two KRDB update mechanisms; evaluation feedback and KRDB synchronization. We also showed the prototype implementation of the proposed P2P search system.

The remaining issues we are now working on include 1) how to protect our system from DoS attacks or malicious users who sends large amount of meaningless queries, 2) how to keep compatibility with original decentralized P2P search systems with no keyword expansion capability, 3) how to find most adequate nodes to synchronize KRDBs with, 4) performance evaluation.

References

[1] Steve Lawrence and C. Lee Giles. Searching the World Wide Web. Science, Vol. 280, no. 5360, pp. 98-100, 1998.

[2] Steve Lawrence and C. Lee Giles. Accessibility of information on the web. Nature, Vol. 400, pp. 107-109, 1999.

[3] Michael K. Bergman. The Deep Web: Surfacing Hidden Value. http://www.brightplanet.com/deepcontent/.

[4] Napster. http://www.napster.com/.

[5] Gnutella. http://gnutella.wego.com/.

[6] Clip2 Distributed Search Services. The Gnutella Protocol Specification v0.4, 2000. http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf.

[7] FastTrack. http://www.fasttrack.nu/.

[8] Qin Lv, Pei Cao, Edith Cohen, Kai Li, and Scott Shenker. Search and Replication in Unstructured Peer-to-Peer Networks. Proc. ACM ICS 2002, June 2002.

[9] Edith Cohen, Amos Fiat, and Haim Kaplan. A Case for Associative Peer to Peer Overlays. Proc. HotNets-I, Oct. 2002.

[10] B. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing. Technical Report, UCB/CSD-01-1141, April 2000.

[11] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. Proc. ACM SIGCOMM 2001, Aug. 2001.

[12] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content-Addressable Network. Proc. ACM SIGCOMM 2001, Aug. 2001.

[13] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. Proc. Middleware 2001, Nov. 2001.

[14] Huebsch, Boon T. Loo, Scott Shenker and Ion Stoica. Complex Queries in DHT-based Peer-to-Peer Networks. Proc. IPTPS 2002, Mar. 2002.

[15] Chunqiang Tang, Zhichen Xu, and Mallik Mahalingam pSearch: Information Retrieval in Structured Overlays. Proc. HotNets-I, Oct. 2002.

[16] Mandar Mitra, Amit Singhal, and Chris Buckley. Improving Automatic Query Expansion. Proc. ACM SIGIR'98, Aug. 1998.

[17] Hersh WR, Price S, Donohoe L, Assessing thesaurus-based query expansion using the UMLS Metathesaurus. Proc. the 2000 Annual AMIA Fall Symposium, 2000.

[18] LimeWire, http://www.limewire.com/.