

Intelligent Knowledge Discovery in Peer-to-Peer File Sharing

Yugyung Lee

School of Interdisciplinary Computing
and Engineering

University of Missouri – Kansas City
leeyu@umkc.edu

Changgyu Oh

School of Interdisciplinary Computing
and Engineering

University of Missouri – Kansas City
Co056@umkc.edu

Eun Kyo Park

School of Interdisciplinary Computing
and Engineering

University of Missouri – Kansas City
ekpark@umkc.edu

ABSTRACT

Emerging peer-to-peer computing provides new possibilities but also challenges for distributed applications. Despite their significant potential, current peer-to-peer networks lack efficient knowledge discovery and management. This paper addresses this deficiency and proposes the Intelligent File Sharing framework, which provides an effective and flexible query for P2P file sharing. The IFS is based on powerful schema and flexible inference, as well as efficiently integrated and extensible retrieval algorithms. Experimental results have provided evidence of the high performance and scalability of the Intelligent File Sharing (IFS) system in peer-to-peer environments.

Categories and Subject Descriptors

H.3.3 [Information Search and Retrieval]: Query formulation, Retrieval models, Search process.

General Terms

Algorithms, Management, Design, Experimentation.

Keywords

Association rules, Encoding, Reasoning, Hierarchy, Peer-to-peer file sharing, Retrieval, Search.

1. INTRODUCTION

There is a growing interest in accessing, relating, and combining data from multiple sources in Peer-to-Peer (P2P) file sharing. Information retrieval is one of the core problems for cooperative file sharing. Many P2P systems [1, 2, 4, 8, 17, 22] have developed mechanisms to achieve desirable P2P applications. Some systems [1, 15] focus on scalable access structures to search for identifiers stored in the distributed directories. Currently P2P file sharing systems provide the file sharing mechanism to peer-end-users with new file searching or managing engines [1, 8]. P2P file sharing applications, like the Gnutella, have become popular as an alternative to the traditional client server systems in some domains

[1]. These systems provide the benefit of storage sharing and serving. These systems are more fault resistant and durable than client-server based systems [3, 17]. These systems are attractive when immediate and temporary information is shared among users. These systems utilize resources which are barely used in client-server based systems.

Although such systems provide many interesting features including a file-sharing method in a decentralized manner, they have a number of the limitations. Large P2P information space raises the issues of complexity and scalability for file sharing. The resources in P2P networks have grown far beyond the complexity that human beings can handle. This creates the problems of searching inefficiency and low information quality. Current P2P file-sharing systems such as Gnutella or Napster [18, 20] are based on file-name search or keyword search. Without an enhanced framework, users find it difficult to effectively retrieve files. It is also hard to obtain files, which are closely related to or required by a file. As a worst scenario, retrieved files become useless due to lack of its application to run with or required files.

Our assumption for this problem is that more intelligent and distributed query mechanisms could increase accuracy and overcome the scalability and performance issues that surpass the current file querying approach. The real time nature of a P2P file sharing application made the development of the P2P challenging as it was necessary to discover relationships and associations of file sharing. For example, if peer hosts know the file association, the peer hosts may send flexible queries to their peer hosts. When the peer hosts download unknown file types, which are contained in a directory such as a music directory or the multimedia directory in a shared directory, the peer hosts may want to know what kind of files they are or which application is needed to open the file. The query can be further refined for finding all the files whose size is smaller than 100 MB and whose type is multimedia (their extension is .avi, .mpeg, or .rm).

In this paper, we demonstrate an intelligent framework called Intelligent File Sharing (IFS) to extend the searching capability of P2P file sharing systems such as Gnutella. The IFS framework efficiently represents not only files to be shared but also their relationships with other files or directories. We specify relationships and schema that are used in the retrieval. A file in the IFS framework is connected to other IFS files, by their index. For instance, a multimedia file can have links to its applications, or to the audio tool files related to it. These links can then be used to discover relationships between IFS resource files using knowledge retrieval and reasoning techniques. They can also be

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

CIKM'02, November 4-9, 2002, McLean, Virginia, USA.
Copyright 2002 ACM 1-58113-492-4/02/0011...\$5.00.

used for an extended search: a user can be interested in the files of a certain type of object, related to another given object. For example, the authors having co-written files with someone, or the files indirectly linked with another file. Based on this knowledge, IFS provides fast reasoning service to find not only resources but also their relationships to the end users. This allows users a more flexible and powerful querying mechanism for P2P file sharing. We will demonstrate how the service gives benefits to P2P file sharing systems.

The rest of paper is organized as follows. In Section 2 we review related works in P2P file sharing. In Section 3 we present a framework that supports P2P file sharing using advanced file relations, their efficient representation and file association algorithms. In Section 4 we describe implementation of the IFS system, experimental results and comparative analysis. We conclude in section 5, where we summarize the main features of the intelligent file sharing.

2. RELATED WORK

Our previous researches [10, 11, 12] focused on efficient domain knowledge management. One popular domain knowledge representation is the concept hierarchy [5, 10, 11]. Hierarchical representation provides methods to convey meaningful context from the lower levels of hierarchy to the higher levels of hierarchy or vice versa. One of the goals of Artificial Intelligence is to provide computer systems with human-like capabilities. In the P2P file sharing system, this goal is also applicable because the peer hosts of P2P file sharing system will have more flexible query options. The reasoning used in the class hierarchies and classification algorithms can also be used to model human-like reasoning [11]. In their papers, classification algorithms have been developed in order to extract interesting patterns and discover meta-rules as a guide for finding multiple-level association rules. These rules can support fast reasoning closure. The IFS adapts its materialized transitive closure to build a fast reasoning hierarchy without using pointer based hierarchical representation.

Meta-search engines can facilitate end users retrieval of information from distributed resources. Meta-search [21] is similar to IFS in the sense that it builds representatives for each database and is an optimizing relationship hierarchy. Current web searching engines have indexer and collection analysis modules in common. They build a variety of indexes on the collected pages and the most common structural information used by search algorithms [3, 9]. Caching Hierarchy [9] tried to construct a monolithic file system indexed by the hierarchical directory tree. Several systems for decentralized lookup services propose scalable access structures to search for identifiers stored in distributed directories [1, 15, 19]. Their approaches differ from our work: First, Gridella [1] uses P-Grid, which is a virtual binary search tree that distributes replicas over a community of peers and supports search. IFS uses file association rules whereas Gridella uses a binary key. Second, Plaxton *et al.* [15] uses a virtual binary search tree as well. Third, Chord [19] is a decentralized lookup service that stores key/value pairs for such networks. CAN [16] uses hash tables for a quick access of P2P resources in large distributed systems. This introduced the concept of a Content-Addressable Network as a distributed infrastructure that provides hash table-like functionality on Internet-like scales.

Browsing Large Digital Library Collections [6] uses classification hierarchies to increase the capabilities of the data browsing in digital libraries. It tries to solve the fuzzy or an incomplete query by using classification hierarchies. Our previous work [12] used the three major relations such as “is-a”, “part-of”, and “contained in”. These three relations define and discover any data relationship of resources in the web sites. Our previous work on efficient transitive closure [10, 11] is very similar to IFS in the sense that class hierarchies are used for inheritance, classification and transitive closure reasoning. Fasttrack [7] provides metadata search. The metadata being detected depends on the file type. Fasttrack is similar to IFS in the sense that meta-data is generated when the file is imported to extract file information. However, IFS uses relationships among files and directories whereas Fasttrack uses Indexing mechanisms for the metadata.

3. IFS: INTELLIGENT FILE SHARING FRAMEWORK

The goal of the IFS framework is to enable powerful schema and flexible inference, as well as efficiently integrated and extensible retrieval of additional tools and to provide an effective and flexible query for file information from heterogeneous P2P sources. For this purpose, the IFS hierarchies are developed to represent file information and their relations. Three relationships (*IS-A*, *Contained-In*, *Run-With*) were introduced. Each file relation describes files related to each other from three different perspectives. The *IS-A* relationship provides specialized file domain knowledge such as “applications and data file type relationship.” The *Run-With* relationship describes “files and their application relationship” (this can be recognized based on file extensions). The *Contained-In* relationship describes “files and their directory relationship.” Thus these relations become the essential basis of the IFS framework for intelligent query.

Our IFS query framework is based on a knowledge base, which is an extension of our previous research, Combined Hierarchical Set (CHS) [12]. The extended CHS called IFS-CHS provides a file association inference mechanism, which is not provided in current P2P applications. IFS-CHS is designed so that the combined relational hierarchies of sharable files can be distributed efficiently and the amount of inter-peer data exchanges required is minimized. The IFS-CHS is a self-contained and order-independent representation. Specifically, with a given IFS-CHS, each peer will maintain its own part of the IFS-CHS according to any changes of its file system, and support interactive queries using IFS-CHS by other peers.

3.1 IFS-CHS Representation

The IFS-CHS representation is based on a directed acyclic graph (DAG). Every node is annotated with data elements, $\langle S, R, C, H \rangle$ consisting of a Set of Number Pairs (*S*), a Relation Type (*R*), Constraint Rule (*C*), and Hierarchy Identifier (*H*). These notations are adapted from our previous work [12].

Definition 1: Set of Number Pairs (*S*): Given a direct acyclic graph G , $S = \{N_1, \dots, N_n\}$, where N is a Number Pair (N) associated with a node A in G and the total number of N is n . The Number Pair (N) is composed of two numbers $[\pi, \mu]$ where $\pi \leq \mu$ and $\pi, \mu > 0$.

1. **The Preorder number π** indicates how many steps the node A was reached in a preorder traversal of the graph G . In the IFS framework π represents file or directory itself.
2. **The Maximum number μ** is the maximum of all the preorder numbers that appear under A . In the IFS framework μ is used to compute how many files or directories exist under the current file or directory by a simple calculation $(\mu - \pi)$.

The number pairs set (S) is used for verifying a transitive reasoning between any pair of nodes in the DAG without actually traversing the path from one node to another node. This encoding allows the IFS system to perform fast transitive closure reasoning and combined association with a set of DAG nodes. DAG links are no longer necessary because the Set of Number Pairs (S) represents all the relationships between the DAG nodes.

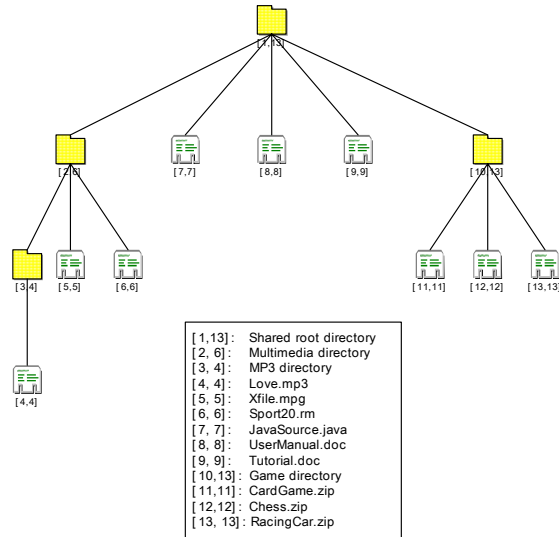


Figure 1. File Representation (Contained-in Hierarchy) using IFS Encoding

Definition 2: Relation Type (R): Our IFS framework provides three relation types: *IS-A*, *Contained-In*, and *Run-With*:

1. ***IS-A*:** This relation describes the inheritance relationship (generalization from general to specific concepts). In the IFS framework, a file extension is used to determine the type of file. The extension is unique and each file can be mapped onto the unique extension. For instant, a movie file, “hackersII.mpg” has file extension “.mpg”. From this example, we define that there is an *IS-A* relation between Movie file and hackersII.mpg (i.e., hackersII.mpg *IS-A* Movie file).
2. ***Contained-In*:** This relation describes that a concept is related to another concept via *Contained-In*. In the IFS framework, the relationship between a directory and the files in the directory can be represented through a *Contained-In* relationship. For example, Acrobat5.0 directory contains three directories, Help, Reader, and Resource. Also the Resource directory contains two directories CMap and Font and a file Enutxt.pdf. Here is a transitive closure through *Contained-In* relationship:

Acrobat5.0 contains Enutxt.pdf because Acrobat5.0 contains Resource and Resource contains Enutxt.pdf.

3. ***Run-With*:** This relation describes how a concept is related to another concept though a special runtime constraint such as A requires B. In the IFS framework, there are two types of files. One is an application type. The other type is a data file for applications. For example, Microsoft windows has a registry for every software package installed in the hard disk. If users double click on a file then the OS links the appropriate application to the file by searching the windows registry file. Zero or more file types may be available for an application. For instance, for Microsoft word.exe, “.doc file”, “.txt file”, or “.dat file” can be an input.

In the IFS framework file hierarchies can be represented in a separate hierarchy with a single relation type or as a combined hierarchy with the three relations. Thus, it is necessary to use explicit notation to avoid confusion between combined relations. Each relation is annotated with a relation type to maintain its identity in the IFS-CHS. The relation types are represented in $R^{s,c,r}$ where s stands for an *IS-A* relation, c *Contained-In*, and r *Run-With*. For a specific relation between files, the following notations are used: $\sigma \xrightarrow{s} \tau$ denotes that τ is a subdirectory file of σ . $\sigma \xrightarrow{c} \tau$ denotes that τ is contained in σ . $\sigma \xrightarrow{r} \tau$ denotes that σ is running with τ . For example, if $\sigma \xrightarrow{s} \tau$ where τ is “.MPG” files and σ is “HACKERSII.MPG” file, then “HACKERSII.MPG” file is a “.MPG” file.

Definition 3: Constraint Rule (C): The rules describe the constraints of file association such as files type, file size and P2P network and peers capacity. The rules are defined as follow:

$$R = Op_1 R_i | R_i Op_2 R_j | R_i Op_3 R_j | R_i R^{s,c,r} R_j | A_i | V_i$$

where A_i is an attribute, V_i is an attribute value, $Op_1 \in \{\text{NOT}\}$, $Op_2 \in \{\text{AND, OR}\}$, $Op_3 \in \{\leq, <, \neq, \equiv, >, \geq\}$ and $R^{s,c,r}$ is the relation type. For example, if a request file (F) is either a movie file or a MP3 file, but not an image file and then it runs with a window’s multimedia player. This file constraint rule can be represented as follow: $R = ((\text{Type}(F) R^s \text{ "movie"}) \text{ OR } (\text{Type}(F) R^s \text{ "MP3"}) \text{ AND NOT } (\text{Type}(F) R^s \text{ "image"}) \text{ AND } (F R^r \text{ "Window Multimedia Player"}))$. Another example is if a peer host wants a file whose size is less than 1 MB and whose download time is less than 10 minutes, the rule can be represented as follows: $R = ((\text{Download-duration}(F) < 10) \text{ AND } (\text{Time-unit} \equiv \text{minute}) \text{ AND } (\text{Size}(F) < 1) \text{ AND } (\text{Size-unit} \equiv \text{MB}))$.

Definition 4: Hierarchy Identifier (H) is used to distinguish hierarchies when multiple hierarchies are involved in inference. For example, “Love.mp3” is represented as [4 4] in the contained-in hierarchy, whose a hierarchy identifier is “contained-in,” in Figure 1 and it is represented as [3 3] in the is-a hierarchy, whose a hierarchy identifier is “is-a”. Then each number pair is represented with its own hierarchy identifier such as s[3 3] and c[4 4].

Within the IFS framework, domain knowledge is stored as a sequence of pairs of numbers. It first builds the concept

hierarchies for directories and files, and converts them into pairs of numbers, which represent the file and its related information. This is an efficient representation of those hierarchies which can improve the performance of file retrieval and updating.

3.2 IFS Association

Reasoning is performed with the three types of relationships of files. This forms the basis of the IFS framework, which aims at scalability, efficiency, and flexibility of P2P file sharing. The mechanism is based on intelligent information retrieval and management schema and powerful reasoning capability. This allows fast and enhanced file retrieval in the P2P file sharing systems. The IFS provides four types of file associations: *Generalized transitive association*, *Aggregated Association*, and *Constraint-based Association*. The enhanced reasoning capability of IFS is based on the representation and relations we introduced in Section 3.1.

Definition 5: IFS Transitive Closure Assume that there are two files (or directory) F_1 and F_2 and their number pairs are $[\pi_1, \mu_1]$ and $[\pi_2, \mu_2]$, respectively. It can be said that there is a transitive closure between F_1 and F_2 iff there is a condition $\pi_1 \prec \pi_2 \leq \mu_2 \leq \mu_1$.

Figure 1 shows that all directories and files are mapped into the set of number pairs. Using this representation we verify in a constant time that “mp3” directory is a subdirectory of “multimedia” directory since the number pair of the “mp3” directory [3 4] is a sub-range of the number pair of the “multimedia” directory [2 6] (i.e., $2 < 3 < 4 < 6$). From the number pair [2 6], we can figure out that the “multimedia” directory has four children (i.e., $6 - 2 = 4$).

Now, we describe the *Generalized transitive association*. This kind of association can extend the capability of query in File sharing applications. We may have a query such as “find a text editor.” We can use a generalized transitive association to answer the query. For example, mypaper.txt is a text file then a text editor can edit this file. The file mypape.txt runs with Notepad. Thus, Notepad is a text editor. Here is a formal statement of the Generalized transitive association:

Theorem 1: Generalized Transitive Association If $(X \xrightarrow{s} Y \ \& \ Y \xrightarrow{r} Z)$, $(X \xrightarrow{r} Y \ \& \ Y \xrightarrow{s} Z)$ or $(X \xrightarrow{s} A \ \& \ A \xrightarrow{r} B \ \& \ B \xrightarrow{s} Z)$, then $X \xrightarrow{r} Z$. Formally, *Generalized transitive association* can be defined: if $(X_1 \xrightarrow{r1} X_m)$ and $(X_m \xrightarrow{r2} X_l)$ and $(X_l \xrightarrow{r1} X_n)$ then $(X_1 \xrightarrow{r1} X_n)$ where $m \neq l \neq n$ and $r1, r2 \in \{s, r\}$.

Proof: These equations hold true in general.

- (1) **Case 1:** $(X \xrightarrow{s} Y)$: A Window multimedia application X is a multimedia application Y . $(Y \xrightarrow{r} Z)$: A Multimedia file Z is running with the Multimedia application Y . Then, $(X \xrightarrow{r} Z)$: the windows multimedia application X runs with the multimedia file Z . Therefore the first equation is true.
- (2) **Case 2:** $(X \xrightarrow{r} Y)$: a MPG file Y is running with a

Window multimedia application Y . $(Y \xrightarrow{s} Z)$: the MPG file Y is a multimedia file Z . Then, $(X \xrightarrow{r} Z)$: the multimedia application X runs with the multimedia file Z . Therefore the second equation is true.

- (3) **Case 3:** $(X \xrightarrow{s} A)$: A window multimedia player X is a multimedia player A . $(A \xrightarrow{r} B)$: a MPG multimedia file B runs with the multimedia player A . $(B \xrightarrow{s} Z)$: the MPG multimedia file B is a multimedia file Z . Therefore, $(X \xrightarrow{r} Z)$: the window multimedia player X runs with the multimedia file Z . Therefore, theorem 1 is true. ■

Definition 6: Aggregated Association If a directory D contains multiple parts (subdirectories or files), namely D_1, D_2, \dots, D_n then we can denote them as $(D \ R^C \ D_i)$ for $1 \leq i \leq n$. There is an aggregated *Contained-In* relationship between D_i and D . The aggregated *Contained-In* relationship is called *Aggregated Association* and the relationship between D and D_i where $1 \leq i \leq n$

can be represented as follow: $D \ R \ \sum_{i=1}^n D_i$. For instance, we

may have a query such as “find the files on CS101 homework.” Without conceptual framework, we may not be able to answer such a query. Using the *Aggregated Association* the files in CS101 homework directory will be retrieved and the aggregated files will be returned to the query.

Definition 7: Constraint-based Association If $X_m R\{s,c,r\} X_n$ and $X_m[c] \rightarrow X_n$ where constraint c is satisfied, then we infer a “constraint-based association” $X_1[c] \mapsto X_n$. The constraint-based association is very useful in the P2P application because certain constraints limit the scope of the search space. For instance, constraints include file size limits, file transferring rates, or files that are stored in the user’s favorite peer hosts. Here is an example of a query with constraint-based association: find a multimedia file where its size is less than 100 Mbytes, it *Run-With* a windows media player and its downloading takes less than less than 30 minutes. Also the IFS may know about that if “Y-File.mpg” is *Contained-In* multimedia directory and its size is 50 Mbytes then “Y-File.mpg” may be downloaded within 30 minutes.

3.3 IFS Association Algorithms

Now we introduce algorithms for the IFS associations: *Generalized transitive association*, *Constraints-based association*, and *aggregated association*. Due to space limitation, we present an algorithm in Table 1, *Combined Association*, which is a basic operation for these three associations (refer to [13] for details). For the Combined Association, given a IFS Combined Hierarchy Set of concept hierarchy H , hierarchy identifier (H_i) , two number pairs of τ, σ , compute inference value of the multiple hierarchies combined association. In this inference, the relation types *IS-A* (S), *Contained-In* (C), and *Run-With* (R) are used.

Table 1. Combined Association Algorithm

```

Combined-Association ( query )
{//Check what kinds of file relationships are contained in a given query
If (it has is-a relationship && contained-in)
{ call contained_in relationship
  aContainedIn = number pairs with the contained_in relationship;
  If( aContainedIn is not null)
    Return not found; // the specified directory does not exist.
  Else // return all shareFiles under the dir
    shareFiles = contained_inSearch(aContainedIn);}
else if ( it has is-a relationship && run-with relationship)
{ filetype = getFileTypeOfQuery(query); // return file extension
  possibleApplicationList = searchInRun_With(filetype);
if (possibleApplicationList is not null)
  return possibleApplicationList;
else return possibleApplication is not found;
...
shareFile[] contained_inSearch(aContainedIn)
{ while( ContainedIn exists)
  { obj = ContainedIn.List.get(i);
  if( obj == a ContainedIn){
    for ( int j = resultFiles.size() - 1; j >= 0; j-- ){
      ShareFile afile = (ShareFile)resultFiles.get( j );
      If ( afile.getContaionedInNumberOne()
        < sf.getContaionedInNumberOne()
        || afile.getContaionedInNumberOne()
        >sf.getContaionedInNumberTwo()){
        resultFiles.remove( j );
        // remained in the list will be returned.} }
      ShareFile[] results = new ShareFile[ resultFiles.size() ];
      return (ShareFile[]) resultFiles.toArray( results );}}
      return new ShareFile[0]; // return empty map for no search result }

```

4. IFS Implementation and Experimental Results

4.1 The IFS System Implementation

The IFS system was implemented using Java language, jdk1.3.1, as an enhancement of the Phex, which is the Gnutella client application available in public [14]. To enhance Gnutella, the IFS implementation includes intelligent file search mechanism (*IS-A*), application search mechanism (*Run-With*), directory search mechanism (*Contained-In*) and a combined search mechanism with constraints. These specialized search mechanisms enable users to save their file look up time and extra time to organize the result files. Figure 2 (in the last page of this paper) is a screen capture of the IFS query interface for application search mechanism, “looking for *applications* for files in *doc* directory”, which is an example of combined relation search (*IS-A*, *Run-With*, *Contained-In*). This enhanced the Gnutella’s query capability.

4.2 Experimental Results

In this paper, we compare our IFS framework to a keyword based file search and evaluate the general efficiency of the systems. Peer-to-peer networks are highly complex and heterogeneous because their size is not limited and the file sharing environment of peers differs greatly. In addition, peers have a dynamic and unpredictable character, as they join and leave frequently. This complexity cannot be mirrored in a computer laboratory without enormous effort. One approach would be to use a simulation, but this prohibits making comparisons with existing products, which require a real network, not a simulation, in order to operate. As our goal was to improve the query searching time of decentralized Gnutella applications, we needed to compare our framework to it. This cannot be done with a simulation. Instead, we used a local network to evaluate the two systems. Although this does not reflect the peer-to-peer complexity, it reveals the general efficiency of the systems.

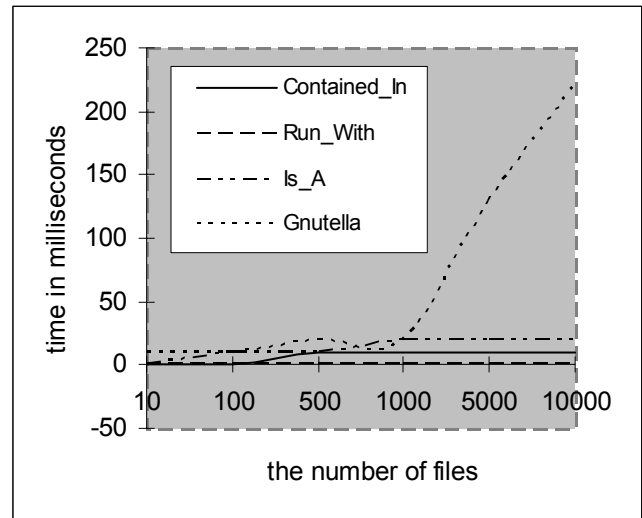


Figure 3. IFS Query Time Comparison with the Gnutella application (Phex)

The tests were performed with two identical computers. Each PC was equipped with a 500 MHz Intel Pentium 3 processor, 128 MB of RAM, and a 100 Mbps Ethernet adapter. Microsoft Windows NT was the operating system. The Java virtual machine was Sun’s Java 1.3.1. All tests were performed with one sender peer and one receiver peer. The performance depends on the total number of files in the shared directory. Each configuration was tested with the number of shared files, which were 100, 500, 1000, 5000, and 10000. Because of the dynamic compilation of the virtual machine, the results required several test runs before they became stable. The presented results shown in Figure 3 are the average of the results of ten tests. This experimental test allowing comparisons between Gnutella and IFS shows the IFS system’s superiority: the required time for query remains constantly regardless the number of the files in IFS while the required time grows exponentially in Gnutella when the number of files is bigger than 5000.

4.3 Comparative Analysis

CAN [16] and Chord [19] are decentralized P2P mechanisms. They are designed to locate named objects. The resources are identified through globally unique names. Thus, to introduce a new resource to the network, the resource must be initialized with a globally unique ID. The Chord protocol has essentially one operation: given a key, it will determine the node responsible for the key. Chord inserts the key/value pairs at carefully chosen nodes. This operation restricts peer hosts to hold only resources with the key/value pairs. Thus Chord is not suitable for hosts which locally update and remove data without any notice to other peer hosts. CAN has the same restriction because CAN uses an indexing mechanism to retrieve a peer host holding requested files. To build the indexing mechanism, internodes operations are needed. Therefore, these systems focus on storage sharing rather than file sharing. If peer hosts do not want to be forced to hold key/value pairs, they cannot join to the P2P networks. Also, if peer hosts in the P2P networks frequently repeat to join and leave, cost for global resource assignment with the key/value pairs will be increased. Napster and Gnutella [18, 20] do not require globally unique resource IDs. It allows peer hosts to join the P2P network without a global configuration. They focus on the file sharing and researchers have described the limited scalabilities of these systems. Napster is based on the traditional client-server concept. A recent report [16] pointed out the inefficiency of Napster's hardware, network bandwidth, or space. Gnutella decentralizes the file location process and increases efficiency. Gnutella application users self-organize into application-level meshes. Requesting messages for a given query are flooded with a certain scope. Flooding on every request is clearly not scalable.

Our framework focuses on file sharing because we do not want to be forced to hold key/value pairs to share resources and do not want to add more computing cost for a global resource configuration. As compared storage sharing systems such as CAN and Chord, IFS resource configuration occurs in a local host. IFS is independent from global internodes configuration. However, other peer hosts, without more computation cost, can use the resource configuration.

Current P2P file-sharing systems such as Gnutella or Napster [18, 20] are based on file-name search or keyword search. Without an enhanced framework, users find it difficult to effectively retrieve files. It is also hard to obtain files, which are closely related to or required by a file. The file association rules in IFS define a file as having *Run-With*, *Contained-In*, and *IS-A* relationships. The relationship helps the system to find not only files but also their information. Any current P2P applications do not provide this kind of mechanism. In Table 2 (in the last page of this paper), we show that the IFS system provides more flexible queries interface for the end users of the P2P file sharing system such as directory search, application search, or combination search with constraints. Flexible search methods save users time by categorizing resources. Users can categorize resources by directory name, application, constraints, etc without any computational overhead. Overall, our IFS framework as shown in Table 2 provides notable advantages, e.g., the ability to enhanced searching capability by using advanced reasoning, the resource availability, and no global initialization for resources.

5. CONCLUSION

This paper describes the P2P File Sharing system based on an intelligent file sharing framework using a fast reasoning approach, which is an enhancement of Gnutella. This framework provides have the breadth users to overcome the limitation of current P2P file sharing systems (i.e., a key based query mechanism). We developed the advanced reasoning approach in IFS. The implementation of our IFS system is examined and the experimental results showed the IFS system's superiority compared to Gnutella. In our demonstration, IFS has more attractive mechanism comparing with keyword search-based applications. Overall, the major contribution of this paper is enhancing scalability and adaptability of current P2P file sharing systems into an intelligent query mechanism.

6. ACKNOWLEDGMENTS

We express our thanks to Jane Vogl for helping us to improve the English in the first version of this paper.

7. REFERENCES

- [1] Aberer, K., Puceva M., Hauswirth, M., and Schmidh, R. Improving Data Access in P2P Systems. IEEE Internet Computing Journal, JAN., 2002, pp.58 - 67
- [2] Anderson, R. *The Eternity service*, in Proceedings of the First International Conference on the Theory and Applications of Cryptology (PRAGOCRYPT '96), Prague, Czech Republic 1996.
- [3] Bolosky, W. Douceur, D. E. and Theimer, M. Feasibility of a Serverless Distributed File System Deployed on an Existing Set of Desktop PCs. In Proceedings of International Conference Measurement and Modeling of Computer System(SIGMETRICS 2000), ACM Press, New York, June 2001, pp. 34 - 43.
- [4] Clarke, I., Sandberg, O., Wiley, B., and Hong, T. Freenet: A Distributed Anonymous Information Storage and Retrieval system. In Proceedings of the Workshop on Design Issues in Anonymity and Unobservability, July 2000, pp. 46 - 66.
- [5] Greiner, R., Grove, A., Roth, D. Learning Active Classifiers. In Proceedings of the Thirteenth International Conference (ICML '96), San Francisco, CA, 1996 pp. 207 - 215.
- [6] Geffner, S., Agrawal, D., Abbadi, A., and Smith, T. Browsing large digital library collections using classification Hierarchies. In Proceedings of the Eighth International Conference on Information Knowledge Management (CIKM99), Kansas City, Missouri, 1999 pp. 195 - 201.
- [7] Harren, M., Joseph, M., Hellerstein, Huebshch, R. Loo, B., Shenker, S., and Stoica, I., Complex Queries in DHT-based Peer-to-Peer Networks. In the First International Workshop on Peer-to-Peer Systems, March 2002.

- [8] Heimbigner, D. Adapting Publish/Subscribe Middleware to Achieve Gnutella-like Functionality. Technical Report CU-CS-909-00, Department of Computer Science, University of Colorado, Sept. 2000.
- [9] Huffaker, B., Jung, J., Wessels, D., and Claiffy K. Visualization of the Growth And Topology of the NLANR Caching Hierarchy. In Proceedings of the Third International WWW Caching Workshop, Manchester, England, June 1998.
- [10] Lee, Y. and Geller, J., Parallel Transitive Reasoning in Mixed Relational Hierarchy, in Proceedings of the Knowledge Representation and Reasoning (KR-96), pp. 576 - 587.
- [11] Lee, Y. and Geller, J, Efficient Transitive Closure Reasoning in a Combined Class/Part/Containment Hierarchy, Journal of Knowledge and Information System, 4(3), pp 305 - 328.
- [12] Lee, Y., Geller, J., Park, E., Oh, C., Data Mining With Distributed Agents In E-Commerce Applications, In Proceedings of the 14th International FLAIRS Conference, AAAI Press, pp. 12-17, 2001.
- [13] Oh, C., Intelligent File Sharing Framework Enhancing Decentralized Peer To Peer Systems Similar To Gnutella, Master Thesis, University of Missouri – Kansas City, 2002.
- [14] Phex: The Gnutella Windows Gnutella Clients, <http://www.gnutelliums.com/windows/#Phex>
- [15] Plaxton, C. G., Rajaraman, R., and Richa, A.W. Accessing Nearby Copies of Replicated Objects in a Distributed Environment. In Proceedings of ACM Symposium on Parallel Algorithms and Architectures, ACM Press, New York, June 1997.
- [16] Ratnasamy, S., Francis, P., Handley, M., Karp, R., and Schenker, S. A Scalable Content-Addressable Network. In Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications August 2001.
- [17] Ripeanu, M., Foster, and Iamnitchi, A, Mapping the Gnutella Network: Properties of Large-Scale Peer-to-Peer Systems and Implications for System Design. IEEE Internet Computing Journal JAN. 2002 pp.50-57.
- [18] Saroiu, S., Gummadi, P. and Gribble, S. A Measurement Study of Peer-to-Peer File Sharing Systems. Multimedia Computing and Networking Conference, San Jose, CA, Jan 2002.
- [19] Stoica, I., Morris, R., Karger D., Kaashoek, M. F. and Balakrishnan, H. Chord: A scalable peer-to-peer lookup service for Internet applications. Technical Report TR-819, MIT, March 2001.
- [20] Tilley, S. and Desouza, M. Spreading knowledge about Gnutella: a Case Study in Understanding Net-Centric Applications In Proceedings of ninth International Workshop on program comprehension, (IWPC 2001) pp. 189 - 198.
- [21] Yu, C. T., Meng, W, Liu, K., Wu, W. and Rishe, N. Efficient and Effective Metasearch for a Large Number of Text Databases. In Proceedings of the Eighth International Conference on Information Knowledge Management (CIKM99), Kansas City, Missouri, 1999 pp. 217- 224.
- [22] Zupan, B., Bohanec, M., Bratko, I., and Cestnik, B. A Dataset Decomposition Approach to Data Mining and Machine Discovery. In Proceedings of the Third International Conference on Knowledge Discovery and Data Mining, Zurich, Switzerland, 1997 pp. 299- 303.

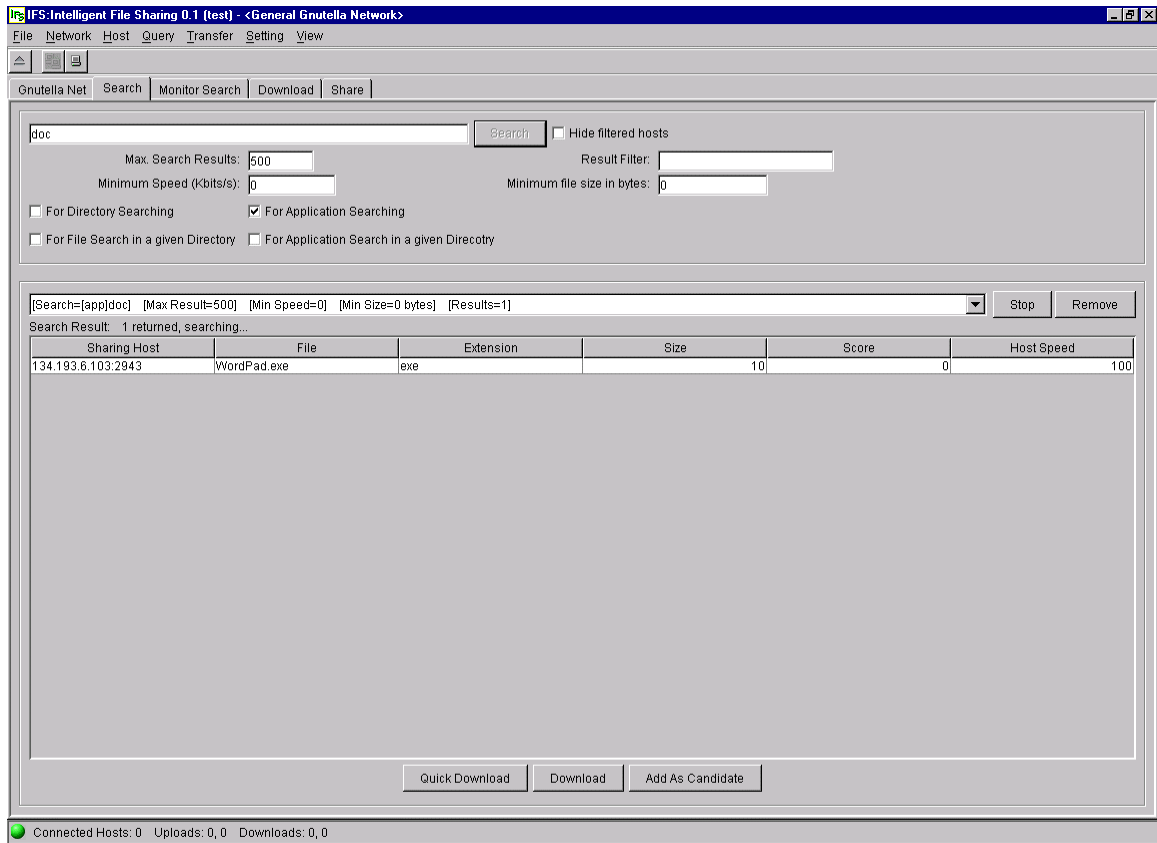


Figure 2. The IFS Query Interface

Table 2. A Comparison of P2P Resource Sharing Systems

Feature/System	CAN [16]	Chord [19]	Napster [18]	Gnutella [20]	IFS
P2P Computing Mode	Decentralized	Decentralized	Centralized + Decentralized (Client Server)	Decentralized	Decentralized
Purpose	Recourse Directory and Storage Services	Recourse Directory and Storage Services	MP3 File Sharing	File Sharing	File Sharing
Resource Representation	Globally Unique IDs (no duplicates)	Globally Unique IDs (no duplicates)	Naming (duplicates allowed)	Naming (duplicates allowed)	File Association Encoding
Preprocess Requirements	Preprocess and interoperable operations for Global IDs required	Preprocess and interoperable operations for Global IDs required	Not required	Not required	Preprocess for Local encoding of files
Query/Searching	ID-based querying and searching using Indexing (key, value)	ID-based querying and searching using Indexing	Keywords	Keywords	Semantic query/searching with advanced encoding and reasoning mechanism