Regular Paper

A Tag-Based Scheme to Realize Real-Time File Search in Hierarchical Peer-to-Peer Systems

Ting Ting Qin[1,a)] Qi Cao[1,b)] Qi Ying Wei[1,c)] Satoshi Fujita[1,d)]

Received: May 20, 2011, Accepted: January 13, 2012

Abstract: In this paper, we propose a new method to realize quick update of information concerned with shared contents in Peer-to-Peer (P2P) networks. The proposed method is a combination of a hierarchical P2P architecture and a tag-based file management scheme. The hierarchical architecture consists of three layers: the top layer consisting of a collection of central servers, the middle layer consisting of a set of sub-servers, and the bottom layer consisting of a number of user peers. Indexes of files held by each user peer are stored at the sub-servers in the middle layer, and the correlation between file indexes and sub-servers is maintained by central servers using tags. We implemented a prototype of the proposed method using Java, and evaluated the performance through simulations using PeerSim 1.0.4. The results of our experiments indicate that the proposed method is a good candidate for "real-time search engines" in P2P systems; e.g., it completes an upload of 10,000 file indexes to the relevant sub-servers in a few minutes and achieves query forwarding to relevant peers within 100 ms.

Keywords: hierarchical P2P architecture, tag-based search algorithm, query forwarding, real-time, sub-server

1. Introduction

Recently, Peer-to-Peer (P2P) systems [23], [25] have attracted considerable attention as a way of providing scalable network services over the Internet. A P2P system consists of a large number of computers called **peers** connected with a logical network called the P2P overlay. Unlike traditional Client/Server (C/S) systems, each peer participating in a P2P can simultaneously play the role of a server and a client, and several services such as file sharing and content delivery are provided among peers in a peer-to-peer manner. A key challenge in P2Ps is how to quickly identify the location of a target file existing in the network since they do not rely on a centralized server as in C/S systems. In the literature, a number of schemes have been proposed to overcome such a difficulty [1], [3], [7], and among those schemes, the use of the *hierarchical* structure is currently the most promising approach to realizing quick access to such location information with a reasonably low cost. Unfortunately, it has not yet been clarified how we can realize *real-time search of update location information* in hierarchical P2Ps, since they lack a mechanism for reflecting dynamic change of location information in the search results.

In the field of Web search, the concept of real-time search emerged around 2007. A typical real-time Web search scheme does not aggregate data to a centralized server like traditional crawler-based schemes since this causes an inevitable delay in reflecting the change of files to the list of collected indexes [24]. Instead, it retrieves data via direct feeds from social websites such as microblogging ones, including Twitter [*1], Jaiku [*2], and more recently Pownce [*3]. Such microblogging certainly fulfills the need for an even faster mode of communication. However, it allows users to represent their status within a very limited number of characters [10], and it can not avoid the return of irrelevant and/or repetitive entries in the search results [6]. As another direction for the study on Web search, tag-based page recommendations such as Social Bookmark Services (SBS) [4], [18], [19] have recently emerged as an attractive way to improve the quality of the Web search (as will be described later, a tag is a keyword or a key phrase whose "meaning" is intuitively understood by the users and could be automatically processed by computer programs). In contrast to full-text search used in crawler-based schemes, tag-based schemes could recommend a Web page relevant to each user by referring to tags attached to each page. It is also pointed out by many researchers that by combining the notion of tags with conventional search engines, we could attain personalization of the search results [20], and achieve semantic (or similarity) search with the notions of ontology and the semantic web [26], [28].

In this paper, we propose a tag-based scheme to realize quick search of update location information in hierarchical P2Ps. The hierarchical architecture consists of three layers; i.e., the correlation between files (held by user peers in the bottom layer) and peers in the middle layer is maintained by the peers in the top layer (in this architecture, a peer in the top layer is referred to as a central server and a peer in the middle layer is referred to as

[1] The authors are with the Department of Information Engineering, Hiroshima University, Higashi-Hiroshima, Hiroshima 739–8511, Japan
[a)] tacit@se.hiroshima-u.ac.jp
[b)] caoqi917@hotmail.com
[c)] weiqypro@hotmail.com
[d)] fujita@se.hiroshima-u.ac.jp

[*1] http://www.twitter.com
[*2] http://www.jaiku.com
[*3] http://www.pownce.com

a sub-server). The search algorithm consists of two parts. The first part determines a way of associating files held by each user peer to sub-servers, where each sub-server has the responsibility to return the search result by referring to the index table as in conventional search engines. The second part provides an efficient way of forwarding a query received by a central server to an appropriate sub-server relevant to the query. We implemented a prototype of the proposed method using Java, and conducted several experiments with one central server, 100 sub-servers and 1,000 user peers. The results of our experiments indicate that it completes an upload of 10,000 file indexes to relevant sub-servers in a few minutes and query forwarding to a relevant peer within 100 ms.

The remainder of this paper is organized as follows. Section 2 gives an overview of related work. Section 3 describes the proposed method. Section 4 discusses about the maintenance of the data structures used in the proposed method. The results of our performance evaluation are given in Section 5. Finally, Section 6 concludes the paper with future works.

2. Related Work

Existing P2P systems can be classified into two types: *unstructured* and *structured* type. Furthermore, structured P2Ps are divided into three types; i.e., centralized, decentralized and hierarchical.

An advantage of unstructured P2Ps is its high flexibility in realizing a complex file search such as similarity search, semantic search, and context search, although it is generally less efficient than structured P2Ps if we restrict our attention to exact matching [11]. Blind search is a basic file search scheme adopted in the original Gnutella [22]. Although it is simple and easily realizable, it causes a number of redundant message transmissions which consume a large amount of network resources. Modified-BFS [14] tries to reduce the number of transmitted messages by bounding the number of receivers for each transmission, which was further restricted to one (except for the transmission from the originator) by using k-random walks [15]. Adaptive Probabilistic Search [27] tries to improve the hit rate of flooding-based schemes by using additional information about the location of target peers. All of the above approaches could certainly improve the performance of the simple blind search used in the original Gnutella. However, they are still insufficient in many P2P applications which require real-time responses.

The file search in structured P2Ps is conducted in a more systematic manner than unstructured P2Ps by using novel data structures such as Distributed Hash Tables (DHTs) and Skip Graphs. Chord is a typical DHT-based P2P [25]. Data allocation in Chord is attained by associating a key with each data item, and by assigning such key/value pairs to the peers through an appropriate hash function applied to the keys. Pastry [23] provides a key-based message routing scheme such that a given message with a designated key is routed to a peer with a unique identifier which is numerically closest to the key. Tapestry provides location-independent message routing to close-by endpoints using only localized resources [29]. An apparent drawback of such hash-based schemes is that two given objects with a high content-similarity will be mapped to random peers. Such a drawback makes it difficult to apply the schemes to a wide class of information retrieval tasks such as similarity and semantic search.

In hierarchical P2Ps such as KaZaa [13] and Morpheus [17], which are both based on the FastTrack protocol [5], peers are automatically selected as superpeers by considering the bandwidth availability and the CPU power. Another self-organizing superpeer system is SOSPNET [8], which is built on top of an unstructured architecture with semantic correlation between peers and files. In this system, superpeers maintain semantic caches of pointers to files, which are dynamically requested by client peers that have similar attributes. It should be noted here that each of the above existing search schemes tightly controls message routing in order to facilitate the search of requested files. Although it would certainly improve the efficiency of the file search process, such a tight control significantly increases the dependency on superpeers, which impacts the fault tolerance and increases the overall cost required for data allocation and topology maintenance.

3. Proposed System

3.1 Underlying P2P Architecture

As mentioned previously, in the proposed system, we adopt a three-tier P2P architecture consisting of top, middle, and bottom layers, where the top layer consists of central servers, the middle layer consists of sub-servers, and the bottom layer consists of a number of user peers (UPs, for short). Let $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$ denote a set of sub-servers. Considering that each central server plays the same role in the proposed system, in the following, we will focus on a central server C as the representative peer. **Figure 1** illustrates an overview of the P2P architecture.

In this system, C takes the responsibility of keeping the correlation between sub-servers and files held by the UPs, and forwarding queries received from UPs to sub-servers relevant to the queries. Each sub-server S_i takes the responsibility of storing file indexes uploaded by UPs; in the following, we say that those UPs are *associated with* sub-server S_i. S_i is also responsible for pro-

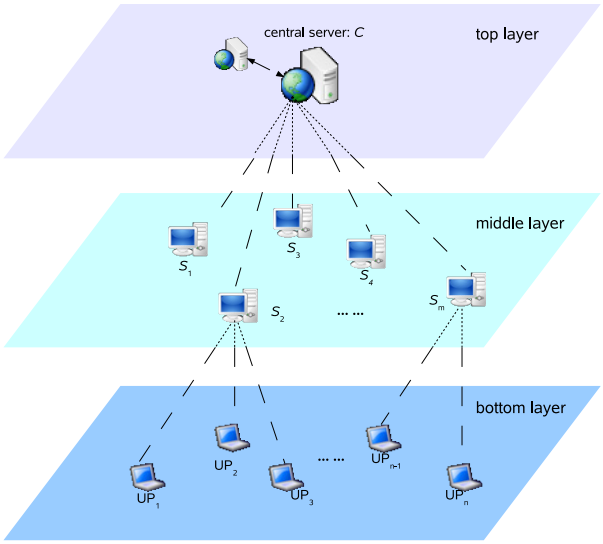

Fig. 1 An overview of 3-tier P2P architecture.

cessing queries received from other peers. Several UPs could be grouped according to the similarity of users' interests behind the peers, and/or the proximity of their geographical locations.

In the following, we describe the details of the proposed scheme. After introducing two basic tools used in the proposed scheme (Section 3.2), we describe a way for the associated sub-servers of gathering file indexes, and a way of quickly identifying sub-servers relevant to a given query (Section 3.3). At last, several issues concerned with those two processes will be discussed in Section 3.4.

3.2 Preliminaries

3.2.1 Tag-Based Sieving of Files

The correlation between sub-servers and file indexes is maintained using the notion of tags. Each tag is a keyword or a key phrase representing the meaning of objects in the real world; e.g., "china" represents several meanings in various contexts including the name of a country and a type of tableware. Let $T = \{t_1, t_2, \ldots, t_n\}$ denote the set of all tags.

T plays two roles in the proposed system. On the one hand, it serves as an index in the file search process and on the other hand, it defines an association of files to the sub-servers. Basically our scheme is proposed for any selection of T. Particularly, to enhance the role of these tags, when determining the set of tags T, we should take into account the popularity of tags, as follows. Zipf's first law, which is a family of related discrete power law probability distributions, states that given a corpus of natural language utterances, the frequency of a word is inversely proportional to its rank in the frequency table [21]. This indicates that we should avoid a selection of high frequency words as a member of T, since it could not attain an efficient sieving of files associated with the tags. Moreover, a tag will not be useful if it is highly unpopular (i.e., if the number of files attached to the tag is quite small), since such an unpopular tag could not reflect the true meaning of the file content. Thus, tags contained in set T must be a low-frequency but a representative word in some sense. In addition, it is well known that in many applications of social tagging systems, users are allowed to add tags for shared information. However, this autonomous modification of tags by participating users does not work well in our proposed scheme. It is because that, in most user-based tagging systems, there is no controlled vocabulary and one authoritative term does not exist to describe a concept or entity. Therefore, it is considered a shortcoming when different users describe objects using many different tags to presumably describe the same thing; e.g., cats, kittens, felines, etc. Because of the characteristic lack of control, there is also no way to regulate the use of singular and plural, which can contribute to navigation difficulties if the system does not include tag stemming when searching. In contrast, expert-added tags can guarantee the precision of representation when describing assets. In this paper, we assume that T is predetermined firstly by several experts or administrators; the proposal of an efficient way of inserting, deleting, and modifying tags in T is left as a future work. One option for maintenance of set T to satisfy the above conditions is to conduct an automatic tag recommendation procedure [2]. With this method, tag sets can be automatically selected from files distributed over the system, and the set of tags can also be periodically updated when updating the files.

Table 1 An example of inclusion relation.

subset	priority sequence	prefix
$T_1 = \{t_2, t_6, t_3\}$	t_2, t_3, t_6	(1) $\emptyset$ (2) t_2 (3) t_2, t_3 (4) t_2, t_3, t_6
$T_2 = \{t_2, t_3\}$	t_2, t_3	(1) $\emptyset$ (2) t_2 (3) t_2, t_3
$T_3 = \{t_5, t_3, t_9\}$	t_3, t_5, t_9	(1) $\emptyset$ (2) t_3 (3) t_3, t_5 (4) t_3, t_5, t_9

3.2.2 Priority Sequence of Tags

Let σ be a bijection from T to $\{1, 2, \ldots, |T|\}$. In the following, $\sigma(t)$ is called the **priority** of tag t, and we say that tag t_1 is given a higher priority than tag t_2 under σ if $\sigma(t_1) < \sigma(t_2)$. Function σ naturally defines the following sequence of tags, which will be referred to as a **priority sequence** of tags in what follows:

$$\sigma^{-1}(1), \sigma^{-1}(2), \ldots, \sigma^{-1}(|T|),$$

where σ^{-1} denotes an inverse of function σ.

We now introduce the notion of the inclusion relation between two tag sets, which plays an important role in the proposed scheme.

Definition 1 Let $T_1, T_2 \subseteq T$ be two subsets of tags. T_1 is said to be included by T_2 under σ, denoted by $T_1 \sqsubseteq_\sigma T_2$, if the priority sequence of T_2 is a prefix of the priority sequence of T_1, where we assume that any T_1 is included by the empty set.

Example 1 Let $T = \{t_1, t_2, \ldots, t_9\}$, and assume $\sigma(t_i) < \sigma(t_{i+1})$ for $1 \le i \le 8$. $T_1 = \{t_2, t_6, t_3\}$ is included by $T_2 = \{t_2, t_3\}$ under σ, since the priority sequence of T_2 is a prefix of the priority sequence of T_1, whereas $T_3 = \{t_5, t_3, t_9\}$ is not included by T_2 under σ. See **Table 1** for an illustration.

Definition 2 Two tag sets T_1 and T_2 ($\subseteq T$) are said to be incomparable under σ, if neither $T_1 \sqsubseteq_\sigma T_2$ nor $T_2 \sqsubseteq_\sigma T_1$.

Function INCLUSION to check the inclusion of T_1 by T_2 is shown in **Fig. 2**.

3.3 Search Algorithm

The proposed method adopts an event-driven approach in the following manner. When a UP joins the system or updates its files, such an event is locally captured by the UP and is notified to the corresponding sub-servers to start collecting the latest updates. Each query which is received by central server C is forwarded to the target sub-server using a similar mechanism, and after receiving the query, the sub-server processes the query, similar to search engines. Namely, it executes a matching process based on its index table which keeps the correspondence between file indexes and UPs, and then directly returns the search result to the requester.

The correlation between files and sub-servers is determined by using the notion of tags as follows. Suppose that each sub-server is associated with a subset of tags, and each file held by a UP is attached to at least one tag by a user. An upload of file in-

function INCLUSION(T_1, T_2)

Step 1: If $|T_1| < |T_2|$, then return false and stop, where $|T|$ denotes the cardinality of set T.

Step 2: If $T_2 = \emptyset$, then return true and stop.

Step 3: Let t_1 be the highest priority tag in T_1, and t_2 be the highest priority tag in T_2. Let $T_1 := T_1 \setminus \{t_1\}$ and $T_2 := T_2 \setminus \{t_2\}$.

Step 4: If $t_1 \neq t_2$, then return false and stop. Otherwise, go to Step 2.

procedure FILE_UPLOAD

Step 1: Let $\hat{T}$ be the set of tags attached to the file index to be uploaded.

Step 2: Find a sub-server S_i which is associated with a tag set T^* such that INCLUSION($\hat{T}, T^*$) is true.

Step 3: Connect to sub-server S_i and upload the file index to S_i.

procedure QUERY_FORWARD

Step 1: Let $\tilde{T}$ be the set of tags corresponding to a query q received by the central server C.

Step 2: C identifies a sub-server S_i which is associated to a tag set T^* such that INCLUSION($\tilde{T}, T^*$) is true.

Step 3: C connects to S_i and forwards q to S_i.

Step 4: After receiving q, S_i conducts a file search similar to conventional search engines, and directly notifies the result to the requesting UP. The number of matching results is notified to C.

Step 5: If the number of matching results is smaller than some predetermined NR, C tries to find another sub-server S_j such that the associated tag set T_j contains at least one of the same tags in $\tilde{T}$, then go to step 3. Otherwise, it stops.

Fig. 2 Procedures used in the proposed algorithm.

dexes to a particular sub-server is conducted by calling procedure FILE_UPLOAD described in Fig. 2, which is based on the check of inclusion relation between two given tag sets. This procedure is invoked by a UP when a file is newly created and/or the content of a file is modified by the UP, and a request of uploading indexes is handled by central server C to determine a sub-server to which the given file index should be transferred.

On the other hand, after receiving a query from a peer, C forwards the query to a sub-server relevant to the query by calling procedure QUERY_FORWARD, described in Fig. 2. The main difference between our three-tier P2P and conventional search engines is that C plays the role of a controller to balance the network traffic in the whole system. In fact, in this procedure, a system variable NR indicating the total number of files discovered so far, plays a similar role to the TTL in flooding-based schemes; i.e., every time a new file is discovered, NR is incremented by one, and the search process stops when NR reaches a predefined value.

3.4 Discrimination Tree and Clusters

In the proposed scheme, each sub-server is associated with a subset of tags in such a way that for any $T' \subseteq T$, there exists a sub-server which is associated with a set of tags including T' under σ considering INCLUSION function. Such an assignment of tags can be graphically represented by a **discrimination tree** in the following manner (note that in the following construction, each tag set is included by exactly one tag set in the resultant tree; we could modify the tree structure in such a way that each tag set is included by several tag sets if we want to increase the robustness of the overall scheme).

Definition 3 (Discrimination Tree) A discrimination tree is defined as follows:

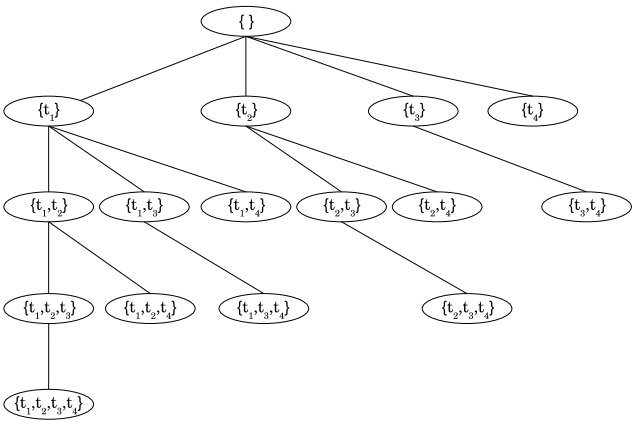

Fig. 3 An example of a discrimination tree.

- Each vertex in the tree is associated with a set of tags, where the root is associated with an empty set of tags. In the following, let $T(u)$ denote a set of tags associated with a vertex u in the tree.
- Let u be a vertex in the tree, and t' be the lowest priority tag in $T(u)$. Let $i' = \sigma^{-1}(t')$ for brevity. Then, in the tree structure, vertex u has no children or it has $n - i'$ children associated with a tag set $T(u) \cup \{\sigma(j)\}$ for each $i' + 1 \leq j \leq n$.
- Each leaf in the tree corresponds to a tag set associated with a sub-server, and a sub-server associated to a leaf plays the role of its parent if it is the left-most child of the parent (such a copy of the "role of parent" is recursively conducted until it reaches the root vertex).

An example of discrimination tree for $T(u) = \{t_1, t_2, t_3, t_4\}$ with $\sigma(t_i) < \sigma(t_{i+1})$ for $1 \leq i \leq 3$, is illustrated in **Fig. 3**. Observe that a collection of resultant tag sets certainly satisfies the requirement described above. Such a tree structure can be used for the "discrimination" of a given query, in a sense that a query received from a peer is placed at the root vertex, and moves toward a leaf vertex associated with a tag set including the query. Thus the time required for determining the relevant sub-server is (almost) proportional to the depth of a leaf vertex relevant to the query.

Given a set of files attached to a set of tags, the above tree structure induces a partition of the set of files such that each subset is associated with a vertex in the tree. More concretely, by associating each vertex to a sub-server (in the hierarchical P2P), we could obtain a clustering of files and peers such that: 1) each cluster is identified by a set of tags, 2) each cluster contains exactly one sub-server, and 3) each file is classified into exactly one cluster (note that we could increase the robustness by associating several sub-servers to a vertex, but such a dependability issue is left as future work). In the implementation, the process of assigning each leaf node in a discrimination tree with a sub-server is executed as follows: 1) We can sort leaves according to the heaviness of the load which is determined by the popularity and frequency of queries associated with the leaves. 2) We could assign leaves to sub-servers so that the (estimated) load of sub-servers is as even as possible. 3) Although the complexity of the problem of distributing given loads to sub-servers so as to minimize the deviation is NP-hard (because it includs partition problem as a special case), experimentally, several heuristic schemes such as a greedy

one is known to exhibit reasonable performance [9]. A key observation we need to notice here is that the size of the clusters severely affects the workload of sub-servers. In the next section, we describe a way of controlling the size of each cluster, as well as a way of realizing dynamic join/leave of participating peers in the proposed scheme.

4. Maintenance Schemes

4.1 Dynamic Join and Leave

In conventional P2Ps such as unstructured P2Ps and P2P DHT, join and leave of peers are realized by refreshing the pointer table maintained by each peer participating in the network. In contrast, the procedure for join/leave is very simple in our proposed method.

To join the system, a new UP first connects to the central server C with its local information such as the attribute and the feature status represented by a collection of tags. C assigns the UP with a sub-server with the highest similarity to the UP, and then the UP establishes a connection to the assigned sub-server (if the number of connections of the sub-server exceeds a predefined value, C should try to find another sub-server). On the other hand, as for the leaving of UPs, we should distinguish the following two cases; i.e., normal leaving and unexpected leaving. In the case of normal leaving, a leaving UP simply sends a request to the sub-server to disconnect it from the network. Meanwhile, the sub-server updates the index information in its index table so that file indexes provided by the leaving peer are completely eliminated. An unexpected leaving of UPs is detected by the corresponding sub-server by periodically transmitting "hello" packets. Upon detecting an unexpected leaving of a UP, it updates the index table as in the case of normal leaving.

Join and leave of sub-servers requires a significant cost compared with the case of UPs. Although it could be achieved by using a similar technique to conventional structured P2Ps, we leave the problem of efficient realization of such a maintenance operation as a future work.

4.2 Cluster Management

In the proposed method, we use the following simple split/merge operations to keep the size of each cluster within an appropriate range. Here the size of a cluster is defined as the number of peers in the cluster. It should be noticed that a user peer can belong to several clusters whereas a sub-server can belong to only one cluster since files owned by the same user peer could be uploaded to different sub-servers by the notion of tags. It should also be clarified that in such a situation when a user peer belongs to several clusters, the amount of duplication of information of the user peer is equals to the number of associated clusters.

Rule 1: Split a cluster into two halves if the size of the cluster exceeds $3x-1$, where x is an appropriate parameter.

Rule 2: Merge two clusters if the size of each cluster becomes smaller than x.

When the size of a cluster becomes $3x$, it is divided into two subclusters of size $3x/2$ each, and the resultant clusters will not be split or merged, until the cluster size increases to $3x$ or decreases to $x-1$. On the other hand, when the size of a cluster becomes $x-1$, it is merged with another cluster whose size is at most $x-1$. The size of the resultant cluster is at least x and at most $2x-2$; i.e., it is (re)merged when the size of the cluster becomes $x-1$ again, and it is split into two subclusters when the size increases to $3x$.

5. Performance Evaluation

5.1 Prototype System

To evaluate the performance of the proposed method, we implemented a prototype of the proposed system. The prototype system is written in Java, and is developed under the following environment: open-SUSE/10.1, Intel Core™ 2 Duo CPU 3.00 GHz, Memory 2 GB, Eclipse/3.4, and JDK/1.6. We used TCP/IP as the underlying communication protocol, and all data including queries and query responses are encapsulated into a message with a header of two bytes in length.

The prototype system uses the following three data structures; i.e., **Global Indexer** in the top layer which keeps the correspondence between tags and sub-servers, **Group Indexer** in the middle layer which keeps the correspondence between files and UPs, as well as the correspondence between keywords and files, and **Local Indexer** in the bottom layer which collects the file contents stored in a UP and generates the index items prepared for Group Indexer.

Concrete software architecture between peers is shown in **Fig. 4** and **Fig. 5**. Figure 4 represents modules used in the prototype system, and Fig. 5 represents the procedure of message transmitting in each peer. More concretely, as shown in Fig. 4, messages are sequentially exchanged among modules in the three layers, and as shown in Fig. 5, each peer has a listener to handle received messages. Each message is received through a message pipe, and pushed into a message queue (arrows (1) and (2)). Then the listener handles those messages one by one (arrow (3)), parses them (arrows (4) and (5)), and sends a new message to other listeners (arrow (6)).

5.2 Setup

We conducted the following two experiments over the prototype system using PeerSim 1.0.4 [12]. In the first experiment, we evaluated the average time required for file upload and query forwarding, as well as the impact of such an update frequency and the scalability with respect to the number of UPs. In addition, we also evaluated the searching efficiency of the prototype sys-

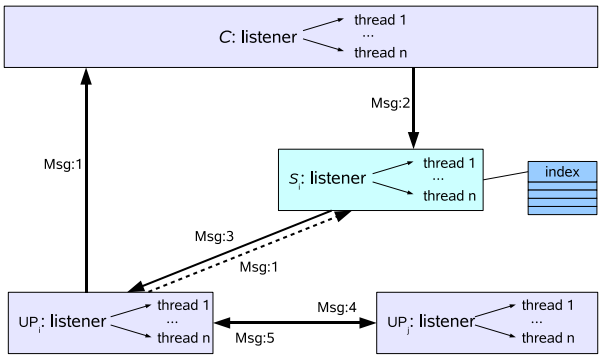

Fig. 4 Modules in three layers.

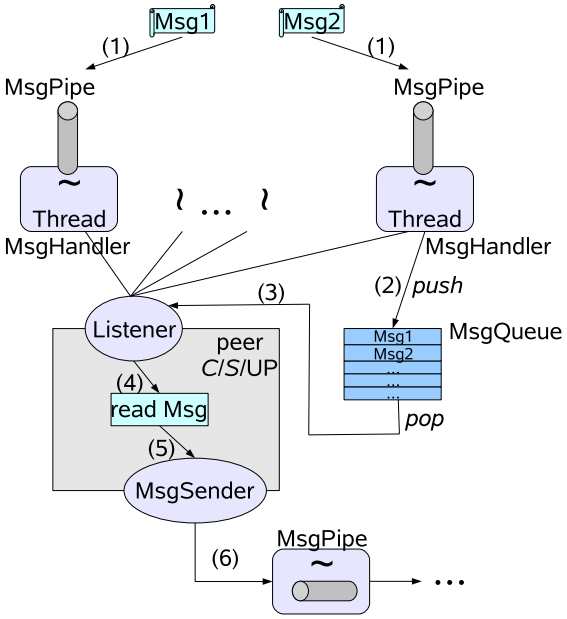

Fig. 5 Message transmitting in each peer.

tem compared with a centralized type P2P which has only one centralized server to control the traffic in the whole network. In the second experiment, we compared the performance of the prototype system with KaZaa [16], in terms of: 1) the success rate of queries, 2) the average search hops of those queries, and 3) its average response time, where the search hop is the number of query forwardings during the query processing (recall that the number of query forwardings is controlled by parameter NR in the proposed system).

It is well-known that the KaZaa overlay is organized in a two-tier hierarchy consisting of superpeers and ordinary peers. KaZaa maintains a file index that maps file identifiers to the IP addresses, which is uploaded from peer to its parent superpeer including the file name, the file size, and the file descriptors. Notice that the file descriptors are used for keyword matches during querying. This file index is distributed across the superpeers. When a user starts a searching activity to locate files, the user's peer sends a query with keywords to its parent superpeer. Then the superpeer returns searching results corresponding to the query. Each superpeer maintains connections with other superpeers, creating an overlay network among the superpeers. Moreover, when a superpeer receives a query, it may forward the query to one or more of the superpeers to which it is connected. In particular, each superpeer also maintains a local index for all of its child peers.

In the simulation of our proposed method, we considered a three-tier P2P system consisting of one central server, 100 sub-servers and 1,000 UPs. Note that in order to make the performance comparison executable, such a selection of the number of sub-servers as well as UPs must be consistent with the setting adopted in KaZaa such that the ordinary peer-to-superpeer ratio is set around 10:1. PeerSim is composed of two simulation engines, i.e., a simplified cycle-based one and an event-based one. We adopt the cycle-based engine in which the cycle time is fixed to 20 s. Each simulation is composed of a number of cycles. During each cycle, each UP independently accesses its protocol such as the file update and the query issue, and such an activity will be

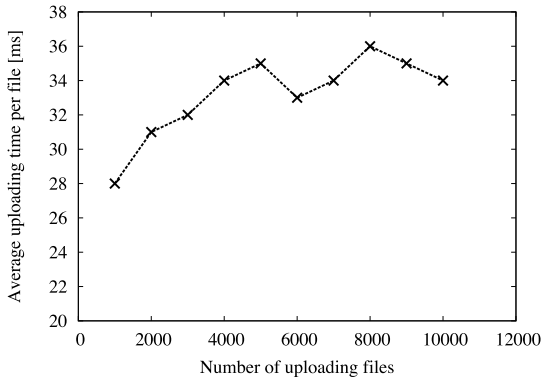

Fig. 6 Average uploading time.

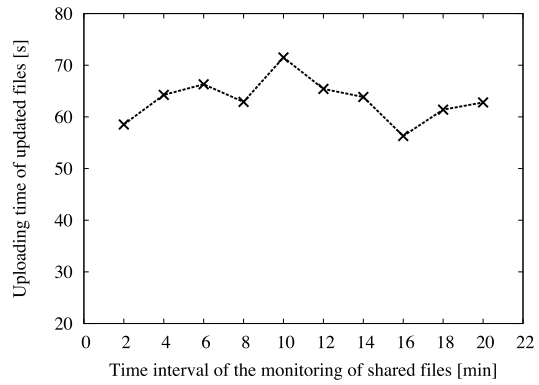

Fig. 7 Uploading time of file index under practical situation.

repeated until the simulation is finished. Meanwhile, the delay of links in the network is classified into four orders of magnitude, i.e., 10 ms, 100 ms, 1 s, or ∞, where ∞ represents the timeout of a connection, and the actual delay of each link is randomly selected from such values. Each UP independently repeats join and leave in a random manner with the average interval time of 20 s, and the update rate of the network state information is fixed to 15 s.

In the simulation, we used T consisting of 100 tags. Each UP has 10 files, and each file is attached to at least one tag according to the Zipf's first law [*4]. Each sub-server is associated with a single tag in T for simplicity. During a simulation, each UP repetitively issues queries associated to at least one tag in T, where the time interval between two successive query processings is fixed to a few seconds. The processing of a query is said to be successful if it identifies the location of the target file. Finally, the value of parameter NR is set to 10.

5.3 Results

Figure 6 shows the average upload time of file indexes, where the horizontal axis is the number of indexes uploaded during a cycle time of PeerSim which is fixed to 20 s as mentioned previously. We could observe that: 1) the average upload time is bounded by 40 ms, while 2) it slightly increases with the number of uploaded indexes per cycle time. This is due to the fact that the process of file uploading is executed completely in parallel. **Figure 7** shows the result under a more practical setting; i.e., 10%

[*4] Under the Zipf's first law, the probability that the i^{th} popular tag t_i is selected as an attachment to a file is proportional to $(1/(i+1))^{\epsilon}$ for some constant ϵ, where the parameter ϵ is generally referred to as Zipf's parameter, and fixed to 1.6 in the current experiment.

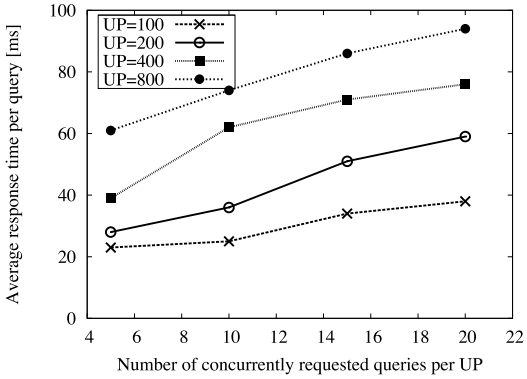

Fig. 8 Average time required for the query forwarding.

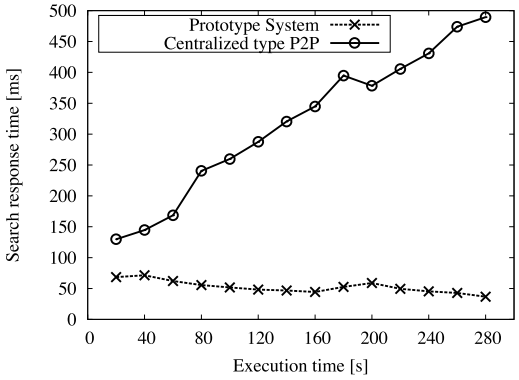

Fig. 9 Search response time.

of file indexes held by each UP are updated per minute (it almost corresponds to a situation in which 1,000 files are updated per minute). The horizontal axis is the time interval of the monitoring of shared files by each UP, which is a function of the prototype system and the time interval is varied from 2 min to 20 min in this experiment. It is worth noting that the prototype system exhibits very nice performance under such a practical situation; i.e., each updating process could be accomplished around one minute, and each file index could be uploaded to a sub-server relevant to the index in around 30 ms.

Next, we evaluate the time required for the query forwarding. **Figure 8** summarizes the result. The horizontal axis of the figure is the number of queries issued by each UP, and each curve is associated to a given number of UPs; i.e., 100, 200, 400 or 800. According to the figure, the average response time is less than 100 ms for all cases, while it becomes worse as the number of issued queries and the number of participating UPs increase; i.e., although the central server becomes a bottleneck for large number of queries, the performance degradation could be bounded by a reasonable value which is fixed to 100 ms in this situation. Another important point we observed from the experiment is that every query is forwarded successfully to its relevant sub-server as long as we neglected the network connection delay or peer unexpected leaves. Next step we will consider multi-match when the sub-server is associated with multiple tags.

Figure 9 shows the result of the response time of searching for the prototype system compared with a centralized type P2P system. In the centralized type P2P system, there is only one central server to take charge of the traffic in the whole network. I is obvious to see that the searching response time increases sharply

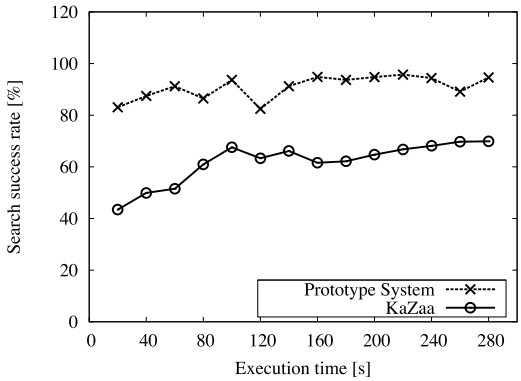

Fig. 10 Search success rate comparison with KaZaa.

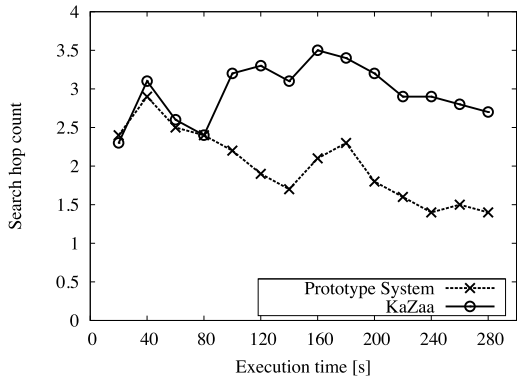

Fig. 11 Search hops comparison with KaZaa.

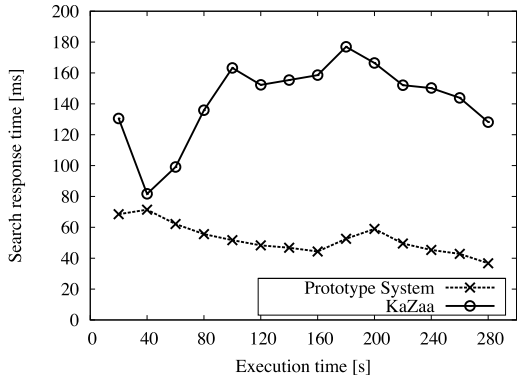

Fig. 12 Search response time comparison with KaZaa.

when compared with the prototype system. This is caused by the sub-servers in the middle layer which share the load from the central server. However, the central server in a centralized type P2P, in contrast, becomes the bottleneck of the whole network traffic.

Finally, we compare the performance of the proposed system with KaZaa with respect to the three metrics described above. The results are summarized in **Fig. 10**, **Fig. 11** and **Fig. 12**. The horizontal axis of the figures is the elapsed time from the start time of the simulation, and the three figures correspond to the three metrics, respectively. Although there is a fluctuation for both schemes in an early stage of the simulation, our system gradually outperforms KaZaa in all metrics, which is the effect of new techniques introduced in the proposed method; i.e., similar file indexes are uploaded to the same sub-server by using the notion of tags, and a given query is immediately forwarded to its relevant sub-servers.

6. Concluding Remarks

In this paper, we proposed a tag-based scheme to achieve quick file searching in P2P environments. To demonstrate the performance of the proposed scheme, we implemented a prototype system and conducted several experiments. The results of our experiments indicate that it could certainly be used as a basic framework for a real-time search in P2P networks in a sense that a query issued by a user will be delivered to its relevant sub-server within 100 ms, an update of files could immediately be reflected the search result, and the workload of sub-servers is bounded by a small value so as to attain high scalability.

Our future work is as follows. First, we should enhance the dependability of the proposed scheme by improving the way tags are associated to files and sub-servers. How to efficiently realize task migration is another important issue. Although the split/merge of clusters could improve the performance of the scheme by bounding the workload of sub-servers within an appropriate range, a migration of clusters (i.e., tasks) causes additional costs which should be reduced by introducing other techniques such as pre-fetching and cache. The development of a way of (automatically) attaching tags to files is also important future work.

Reference

[1] Aspnes, J. and Shah, G.: Skip graphs, *ACM Trans. Algorithms* (*TALG*), Vol.3, No.4 (Nov. 2007).
[2] Ang, H.H., Gopalkrishnan, V., Ag, W.K. and Hoi, S.C.H.: P2PDocTagger: Content management through automated P2P collaborative tagging, *Proc. 36th International Conference on VLDB*, Vol.3 (Sep. 2010).
[3] Broder, A. and Mitzenmacher, M.: Network applications of bloom filters: A survey, *Internet Mathematics*, Vol.1, No.4, pp.485–509 (2004).
[4] available from ⟨http://delicious.com/⟩ (2010).
[5] available from ⟨http://www.fasttrack.nu/⟩ (2009).
[6] Geer, D.: Is it really time for real-time search? *Computer*, Vol.43, No.3, pp.16–19 (Mar. 2010).
[7] Gribble, S., Brewer, E., Hellerstein, J. and Culler, D.: Scalable, distributed data structures for internet service construction, *Proc. 4th Conference on Symposium on Operating System Design and Implementation* (*OSDI*), Vol.4, p.22 (Oct. 2000).
[8] Garbacki, P., Epema, D.H.J. and van Steen, M.: The design and evaluation of a self-organizing superpeer network, *IEEE Trans. Comput.*, Vol.59, No.3, pp.317–331 (Mar. 2010).
[9] Godfrey, B., Lakshminarayanan, K., Surana, S., Karp, R. and Stoica, I.: Load Balancing in Dynamic Structured P2P Systems, *Proc. IEEE INFOCOM*, Vol.4, pp.2253–2262 (Mar. 2004).
[10] Huberman, B.A., Romero, D.M. and Wu, F.: Social networks that matter: Twitter under the microscope, *First Monday*, Vol.14, No.1-5 (Jan. 2009).
[11] Harren, M., Hellerstein, J.M., Huebsch, R., Loo, B.T., Shenker, S. and Stoica, I.: Complex queries in DHT-based peer-to-peer networks, *Proc. 1st International Workshop on Peer-to-Peer Systems* (*IPTPS*) (Mar. 2002).
[12] Jelasity, M., Montresor, A., Jesi, G.P. and Voulgaris, S.: The PeerSim simulator, available from ⟨http://peersim.sf.net/⟩ (2009).
[13] available from ⟨http://www.kazaa.com/⟩ (2009).
[14] Kalogeraki, V., Gunopulos, D. and Zeinalipour-Yazti, D.: A local search mechanism for peer-to-peer networks, *Proc. 11th International Conference on Information and Knowledge Management* (*CIKM*), pp.300–307 (2002).
[15] Lv, Q., Cao, P., Cohen, E., Li, K. and Shenker, S.: Search and replication in unstructured peer-to-peer Networks, *Proc. 16th International Conference on Supercomputing*, pp.84–95 (2002).
[16] Liang, J., Kumar, R. and Ross, K.: The Kazaa overlay: A measurement study, *Proc. 19th IEEE Annual Computer Communications Workshop* (2004).
[17] available from ⟨http://www.morpheussoftware.net/⟩ (2010).
[18] Mathes, A.: Folksonomies-cooperative classification and communication through shared metadata, available from ⟨http://www.adammathes.com/academic/computer-media-ted-communication/folksonomies.html⟩ (Dec. 2004).
[19] available from ⟨http://myweb2.search.yahoo.com/⟩ (2010).
[20] Noll, M.G. and Meinel, C.: Web search personalization via social bookmarking and tagging, *Proc. 6th International Semantic Web Conference and 2nd Asian Semantic Web Conference* (*ISWC/ASWC2007*), pp.365–378 (Nov. 2007).
[21] Newman, M.: Power laws, pareto distributions and Zipf's law, *Contemporary Physics*, Vol.46, pp.323–351 (2005).
[22] Ripeanu, M.: Peer-to-Peer architecture case study: Gnutella network, *Proc. 1st International Conference on Peer-to-Peer Computing* (*P2P'01*) (Aug. 2001).
[23] Rowstron, A.I.T. and Druschel, P.: Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems, *Proc. IFIP/ACM International Conference on Distributed Systems Platforms* (*Middleware*), pp.329–350 (Nov. 2001).
[24] Spark, D.: Real-Time search and discovery of the social web Spark Media Solution (Dec. 2009).
[25] Sotica, I., Morris, R., Karger, D., Kaashoek, M.F. and Balakrishnan, H.: Chord: A scalable peer-to-peer lookup protocol for internet applications, *IEEE/ACM Trans. Net.*, Vol.11, No.1, pp.17–32 (Feb. 2003).
[26] Tran, T., Cimiano, P., Rudolph, S. and Studer, R.: Ontology-based interpretation of keywords for semantic search, *Proc. 6th International Semantic Web Conference and the 2nd Asian Semantic Web Conference* (*ISWC/ASWC2007*), pp.523–536 (Nov. 2007).
[27] Tsoumakos, D. and Roussopoulos, N.: Adaptive probabilistic search for peer-to-peer Networks, *Proc. 3rd International Conference on Peer-to-Peer Computing* (*P2P'03*) (Sep. 2003).
[28] Zezula, P., Amato, G., Dohnal, V. and Batko, M.: Similarity search: The metric space approach, *Advances in Database Systems*, Vol.32, Springer-Verlag (2006).
[29] Zhao, B.Y., Huang, L., Stribling, J., Rhea, S.C., Joseph, A.D. and Kubiatowicz, J.D.: Tapestry: A resilient global-scale overlay for service deployment, *IEEE Journal on Selected Areas in Communications*, Vol.22, No.1, pp.41–53 (Jan. 2004).

Ting Ting Qin received her B.E. degree and M.E. degrees in computer science from Jilin University, China, in 2005 and 2008 respectively. She is currently a Ph.D. student at the Graduate School of Engineering, Hiroshima University. Her research interests include the design, implementation and performance analysis of peer-to-peer systems. She is a graduate student member of IEEE, IEEE Computer Society, and IEEE Women in Engineering.

Qi Cao received his B.E. degree from Tianjin University of Science and Technology, China, in 2007 and his M.E. degree from Hiroshima University, Japan, in 2010. He is a Ph.D. student at the Department of Information Engineering, Hiroshima University. His research interests include modeling, analysis and performance evaluation of peer-to-peer systems.

Qi Ying Wei received his B.E. degree in software engineering from Chongqing University, China, in 2008 and M.E. degree from Hiroshima University in 2011. His research interests include P2P networks and information retrieval.

Satoshi Fujita received his B.E. degree in electrical engineering, M.E. degree in systems engineering, and Dr.E. degree in information engineering from Hiroshima University in 1985, 1987, and 1990, respectively. He is a Professor at the Faculty of Engineering, Hiroshima University. His research interests include communication algorithms on interconnection networks, parallel algorithms, graph algorithms, and parallel and distributed computer systems. He is a member of IEICE, SIAM Japan, IEEE, and SIAM.