Efficient File Sharing Strategy in DHT Based P2P Systems

Zhiyong Xu
Suffolk University
zxu@mcs.suffolk.edu

Xubin He
Tennessee Tech University
hexb@tntech.edu

Laxmi Bhuyan
University of California, Riverside
bhuyan@cs.ucr.edu

***Abstract*—**
In Peer-to-Peer (P2P) file sharing systems, the participating peers share the files with others. Two steps are needed for file sharing: First, a routing request is generated and sent to other peers by using a routing algorithm. The feedback received by the client contains the location information of the requested files; Second, the client retrieves the file from one or more peers which have a copy of that file. Routing algorithms have great impact on the overall system performance, distributed Hash Table (DHT) based routing algorithms provide an elegant and efficient mechanism and become popular in recent years. However, two problems exist in DHT algorithms. First, to find out the location information, in some cases, the routing request may traverse distant peers around the world; Second, in case of multiple copies of the requested file stored on different peers, there's no way to figure out which peer is the topologically closest to the client. Thus, the client may have to download the file from a remote peer and suffer from long retrieve latency. In this paper, we propose a hierarchical routing and retrieving algorithm to relieve these problems. The peers' topological information is utilized. Our algorithm is able to find out the closest copy for any routing requests. The simulation results show our strategy can significantly improve the system routing and retrieval performance.

I. INTRODUCTION

In recent years, Peer-to-Peer (P2P) systems attracted a great deal of attention from both academic and industry communities. Unlike the traditional Client/Server architecture, in P2P systems, there's no clear separation between clients and servers. All the peers share their resources and work cooperatively to provide the service. P2P applications can be characterized as distributed systems in which all the peers have the identical responsibility and all the communications are symmetric. The features of decentralized control, self-organization, fault tolerance and load balancing make P2P systems very attractive for certain environments. P2P applications became popularized through the file sharing applications such as Napster, Gnutella, Kazaa and BitTorrent. [1], [2], [3], [4].

In P2P systems (we use P2P systems to denote P2P file sharing systems), the peers share their files with others. Two operations are necessary when a client (We use client to denote the peer who initiate a file request) wants to retrieve a file. The first one is the routing operation. The client generates a routing request and sends to other peers according to the underlying routing algorithm. After this operation is finished, it will get the location information of the requested file. The second operation is the real file retrieving operation. It will communicate with one or more peers which have that specific file to start a download process. Clearly, efficient routing and retrieving algorithms are essential to achieve high system performance.

Napster uses a central facility to solve the routing problem. The central facility keeps the location information of all the files in the system. As a peer joins the system, the location information of shared files on its storage is published on the central facilities. All the clients send their routing requests to the central facility to get the file location information. Only the retrieving processes are executed among the peers. Some P2P applications use another approach. Gnutella uses a flood-based routing algorithm. A client sends a routing request to all its neighbor peers to check if they have the requested file. If not, these neighbor peers will forward this routing request to their neighbors, until eventually, the file is found.

The central facility used in Napster creates a single point of failure and hotspot issue as in C/S systems, while the flood-based routing algorithm used in Gnutella generate large amounts of unnecessary network traffic. Numerous research projects have been conducted to address these issues. Distributed Hash Table (DHT) based (or structured) routing algorithms such as Chord [5], Pastry [6], Tapestry [7] and CAN [8] have been proposed. They provide an efficient organization infrastructure for P2P systems. In DHT algorithms, each peer is assigned a unique identifier (nodeid) on a large name space (typically, 2^n, n equals to 128, 160 or 192 bits). Each file is also assigned a unique identifier (fileid) on the same name space. The location information of a file is stored on the peer whose nodeid is the numerically closest to the corresponding fileid. A routing process is converted into the process of finding the peer whose nodeid is numerically closest to the requested fileid.

In DHT systems, the location information of all the files are almost equally distributed on the peers, each peer only keep 1/K (K is the total number of shared files) of the location information. A routing request is guaranteed to be finished within log(N) steps. DHT algorithms make good compromise between the information to be stored on a single peer and the routing overhead. It is suitable for large-scale P2P systems with hundreds of thousands or even millions of peers. Many experimental P2P applications using DHT routing algorithms have been developed, such as Oceanstore [9], PAST [10], CFS [11] etc.

The proposed DHT routing algorithms focused on generating a logically elaborated routing mechanism for large scale P2P system with thousands or even millions of computers. These algorithms work well for the ideal environment: symmetric network topology, identical system nodes and uniform workloads. However, they fail to achieve the optimal performance under the real world environment which has complex network connections, diverse peers and various service requirements. One serious issue is the mismatching between the logical routing structure and the physical distribution of peers. The nodeid generated for a peer can not reflect its topological character. Bad effects occurred with such a strategy:

First, unexpected long access latency for some routing requests may occur. In extreme cases, all the routing hops for a request may occur between two peers who are topologically separated and result in long routing latency;

Second, a routing request may need a large number of routing hops to finish. This is because, in DHT algorithms, all the requests must be sent to the peer whose nodeid is the numerically closest to the fileid;

Third, a peer may have to retrieve the requested file from a remote peer even if a near-by peer has a copy. For example, assume there are multiple copies of a file exist on different peers, a client generates a routing request for this file. The peer who is responsible for storing the location information of this file does not know the distance between the client and the peers who have a copy. Thus, it has to randomly choose one from them. Even if it returns all the peers' information to the client, the client still does not know which peer is the closest one, and it may still choose a remote peer and send the retrieving request.

The above problems are caused by the neglect of the topological information in DHT routing structure. In this paper, we introduce a new DHT based P2P algorithm to solve these problems. We use distributed binning scheme to figure out the topological information of each peer, and add this information to the routing and retrieving data structures. We create a hierarchical architecture by generating multiple P2P circles in different layers. The topologically close peers are grouped together in a single circle. A routing request begins at the lowest layer circle and moves up. It can be finished in any layers as soon as the location information of the requested file is found. Thus, the routing overhead is reduced. Furthermore, by adding the peers' topological characteristics to the file location information, our algorithm can guarantee the closest copy of a requested file to be found. The retrieving problem in DHT algorithms is solved.

The rest of the paper is organized as follows, we give the brief introduction of a popular DHT algorithm – Chord, and explain its routing/retrieving problems in Section II. We present a simple solution to solve the retrieving problem in Section III. We describe our topologically-aware routing and retrieving algorithm in Section IV. In Section V, we evaluate the efficiency of our algorithms and compare with Chord. We describe the related works in Section VI and finally, we give out the conclusion and the future work in Section VII.

II. Chord Routing Algorithm

Our algorithm aims to improve the routing/retrieval performance in DHT systems. We use the current DHT algorithms as the underlying algorithm and build our algorithm on top. In this paper, we choose Chord as the underlying algorithm. However, we can build our algorithm on top of any other DHT algorithms as well. In this section, we give a brief introduction of Chord, and describe its problems.

A. The Base Chord Protocol

In Chord, to uniquely identify a peer, consistent hashing [12] is used. An n-bit identifier (nodeid) is assigned to each node on the circular name space $[0, 2^n]$. A collision free algorithm such as SHA-1 [13] is used to generate identifies to avoid the possible duplication problem. The nodeid represents a peer's numerical position on the name space. Each file is also assigned a fileid with the same algorithm. Chord uses *finger tables* to store the information of other peers. On peer *R*, a finger table has (at most) n entries, with the i^{th} entry contains the nodeid and IP

Finger Table

start	intervals	Successor
122	[122,123)	124
123	[123,125)	124
125	[125,129)	131
129	[129,137)	131
137	[137,153)	139
153	[153,185)	158
185	[185,249)	192
249	[249,121)	253

File Location Table

Name	Fileid	Nodeid	NodeIP
File1	102	215	X.X.X.X
File2	107	131	X.X.X.X
File2	107	69	X.X.X.X
File2	107	237	X.X.X.X
⋮	⋮	⋮	⋮
FileK	121	214	X.X.X.X

Fig. 1. Sample finger table and file location table for the Peer 121 on name space [0, 256)

address information of the first peer, S, that succeeds R by at least 2^{i-1} on the name space. It is denoted as R.finger[i].node. In Chord, when a peer joins the system, its finger table is created with all the peers can be used as candidates. Clearly, only the numerical property is considered for the finger table construction. Though not specified directly, Chord has another table: *file location table* on each peer to store the file location information of the files whose fileids are numerically closest to this peer's nodeid. An entry in this table has a file name, its fileid, and the nodeid and IP addresses of a peer who has a copy of this file.

Figure 1 shows the sample finger table and file location table of the peer (nodeid: 121). If node 121 sends a request for a file with the fileid 168, the first routing hop goes to node 158 who is responsible for the interval [153, 185]. If another peer whose nodeid is numerically closer to the fileid 168 exists, node 158 forwards the request to that peer according to its finger table. The process continues until eventually, the request arrives the destination peer whose node is the numerically closest than all the other peers. A request for a file with the fileid 102, 107 or 121 from any peers will reach node 121 after several hops, and it will check the file location table and send the corresponding information back to the clients. More details of Chord protocol can be found in [5].

B. Routing and Retrieving Problems

In Chord, the routing data structures record the peers' numerical characteristics only, the network topological information is not well considered. It is very likely that a routing hop is taken between two peers which are topologically separated. For example, as shown in Figure 2, Client A in Cincinnati, OH may traverse distant peers in Europe, Asia and Africa before its routing request reaches the destination: Client E in Columbus, OH. In this extreme situation, the resulting routing latency may be tens of times higher than the actual distance between Client A and E. However, there's no way to detect and solve this problem in Chord.

Chord also has the retrieving problem. For example, if the file requested by A has copies on Client C, J and X. Although C is much closer than the other two peers, A does not aware of that, it may choose X instead of C to start the download process and result in longer retrieving time.

The reason for the routing/retrieving problems is: we generate the nodeid for each peer using SHA-1 which can not reflect the peer's topological characteristic. A mismatching between the system logical organization and the peers' topological distribution occurs. For example, two peers with the adjacent nodeids could be located in different continents, while two other peers which are topologically close may have separated numerical nodeids. To achieve better performance, we have to add topological information into account when we make the routing

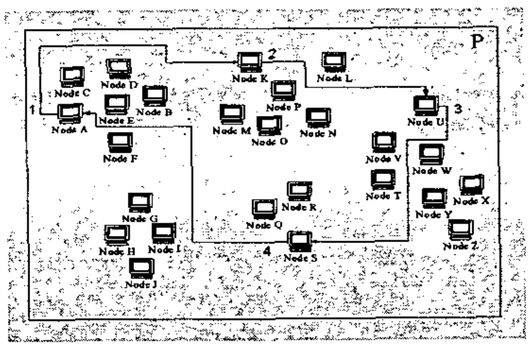

Fig. 2. A sample routing procedure in Chord, the routing request is denoted by arrows. The number of routing hops is 4.

and retrieving decisions. In the following sections, we discuss the solutions to address these problems. We do not want to design a completely new routing algorithm since DHT algorithms are elegant, logically elaborated. We concentrate on improving the performance on current DHT algorithm by adding topological features. With this strategy, we can still use the well designed data structure and routing mechanisms in current DHT algorithms.

III. Utilizing Topological Information to Improve Retrieval Performance

First, we propose a simple solution to address the retrieving problem. Unlike the previous DHT design, in our system, the peers' topological information is used to make the retrieve decision. This information is stored together with the file location information. After receiving a routing request, system will reply the client with the nodeid and IP address information of the peer who is the topologically closest to it. Then, the client can download the requested file from that peer with minimal link latency.

A. *Distributed Binning Scheme*

To choose the topologically closest peer, system should be able to estimate the distance between peers. Thus, we have to use some mechanisms to figure out the approximate topological distribution of the peers. A simple mechanism to do this is the distributed binning scheme proposed by Ratnasamy and Shenker [14]. It uses link latency as the metric to measure the distance between two machines. In this scheme, a well-known set of machines are chosen as the landmark nodes, and they are evenly distributed in the system to ensure accuracy. If k landmark nodes L1, L2, ... , Lk are used, when a peer joins the system, it measures and records the latencies to these nodes in the order of 1, 2, ..., k. Then we can generate an ordering link latency information (denote as order information thereafter) for this newly joined peer. The range of the possible latencies between each [peer, landmark node] pair can be divided into different levels. For example, we can define three levels: level 0 represents the link latencies within [0,20ms], level 1 for the latencies within [20,100ms] and level 2 for the latencies greater than 100ms. Thus, we can use a single digit with the value of 0, 1 or 2 to represent the distance to a landmark node. For k landmark nodes, on each peer, we can use k digits to represent the order information. This information can be viewed as the proximate topological position of a peer in a k-dimensional space. For any two peers, the more number of digits are in common (which means they have similar link latencies to more landmark nodes), the topologically closer these two peers are. The detailed information about the distributed binning scheme can be seen in [14].

We use the order information as part of the node identification information. For example, Node 121:012 represents the peer with nodeid 121. Three landmark nodes (L1, L2 and L3) are used in the system, and the link latencies from this peer to L1, L2 and L3 are within [0,20), [20,100) and greater than 100ms, respectively.

B. *The Simple Retrieving Solution*

With the addition of the order information, the previous retrieving problem can be easily solved. In our simple solution, when a client generates a routing request, it sends its order information as part of the request. After the peer who is responsible for storing the file location information of that particular file receives the request, it will use the client's order information and compare it with order information of all the peers who have a copy of that file. It chooses the peer which has the most number in common with the client in their order information, and returns the nodeid and IP address information of that peer to the client as the answer of the routing request. After receiving the feedback, the client is able to download the file from the topologically closest peer, thus, the retrieve overhead is reduced.

To implement this functionality, we have to modify the structure of the finger table and the file location table in Chord. The sample tables for node 121: 012 are shown in Figure 3. We do not change the routing algorithm. In this case, if peer 45: 020 requests a file with the fileid 107, the routing request will eventually arrive peer 121: 012 using the Chord algorithm. Peer 121: 012 checks order information of the client 45: 020 and compares it with the peers 131: 112, 69: 012 and 237: 020 which have a copy of that file. Clearly, peer 237: 020 is the topologically closest to the client 45: 020 since they have the same order information. Then, peer 121: 012 sends the nodeid and IP address information of 237: 020 as the feedback to the client 45: 020. This client can download the file from peer 237: 020, and avoid retrieve the file from distant peers 131: 112 or 69: 012. Thus, we can achieve satisfactory retrieval performance.

IV. Topological-Aware Routing and Retrieving

A. *The Remaining Problem*

The simple solution can solve the retrieving problem completely. However, the routing issue remains untouched. In both Chord and the simple solution, when a client requests a file, even if there's a copy of that file on a peer who is adjacent to the client, the client has no way to aware of this fact until the routing request reaches the peer who is responsible for this particular fileid. Thus, the mismatching problem of DHT system logical organization and the physical structure still exist. A routing request may traverse several distant peers before reaching the final destination and result in significant routing overhead. Since routing procedures are the most frequently executed operations in P2P file sharing systems, this problem can greatly affect the system efficiency.

This problem is caused by the file location information lookup and maintenance strategy used in DHT algorithms. Only the peer whose nodeid is the numerically closest to a fileid is responsible for storing the location information of that particular file. Thus, all the routing requests for that file must go to this peer, no matter the client is close or far away from this peer. Such a routing strategy is not efficient. We introduce hierarchical architecture to relieve this problem. We take advantages of

Finger Table

start	intervals	Successor
122	[122,123)	124: 011
123	[123,125)	124: 011
125	[125,129)	131: 112
129	[129,137)	131: 112
137	[137,153)	139: 022
153	[153,185)	158: 012
185	[185,249)	192: 001
249	[249,121)	253: 012

File Location Table

Name	Fileid	Nodeid	NodeIP
File1	102	215: 210	X.X.X.X
File2	107	131: 112	X.X.X.X
File2	107	69: 012	X.X.X.X
File2	107	237: 020	X.X.X.X
⋮	⋮	⋮	⋮
FileK	121	214: 111	X.X.X.X

Fig. 3. The sample fi nger table and fi le location table of Node 121: 012 in the simple solution

the peers' topological characteristics to build a hierarchical P2P infrastructure. In our system, topologically close peers are organized as small groups. For each file, multiple peers in different groups are used to store part of the location information. If a client requested file has a copy in a near-by peer, then this location information will be stored on another near-by peer. The client can get the file location information from it and avoid the access to the remote peer.

B. Hierarchical Architecture

We define a P2P *circle* as a collection of peers with related routing and file location data structure. A P2P circle is a self-organized and relatively independent unit, the members in a P2P circle are equally important and take the equal responsibility for the workloads within this circle. In current DHT systems, there's only one P2P circle exist, it contains all the peers. In our system, we create a hierarchical P2P infrastructure: Besides the biggest circle which consists of all the peers, many other P2P circles in different layers which contain different number of peers are generated as well. These circles are created in such a mechanism: the lower the layer, the closer the peers in this circle. Thus, the average link latency between two peers in a lower layer circle is much smaller than the peers in a higher layer circle. We define the number of layers as the *hierarchy depth*. In a *m*-depth P2P system, each peer belongs to m P2P circles with one in each layer. Clearly, the lowest layer circles consist of the set of peers which are the topologically closest to each other.

A simple illustration of a two-layer hierarchical P2P organization is shown in Figure 4. P is the Layer-1 (biggest) circle which contains all the peers. This global layer circle has five layer-2 circles: P1, P2, P3, P4 and P5. Each peer in the system belongs to the layer-1 circle P and one of the five layer-2 circles. For example, Client A is a member of circle P1 and Client Z is a member of circle P5. At the same time, both of them are also members of the global circle P. Topologically adjacent peers are grouped in the same layer-2 circle. For example, Client A, B, C, D, E and F are located on the same continent, they are grouped together in layer-2 circle P1.

In our system, we use the peers' order information to generate circles. For example, if we want to create a two-layer P2P system, we can group peers which have the same order information into one lower layer P2P circle. Thus, Node 121: 012 is in a circle with all the members have the ordering information – 012. We denote "012" as the *circleid* for this circle.

C. Data Structure

To support the routing and retrieving operations within circles in different layers, we have to modify the structure of the finger table. If we create an m-depth P2P system, for each peer, m-layer tables are needed with one in each layer. Figure 5 shows a two-layer finger table for node 121: 012. In the higher layer

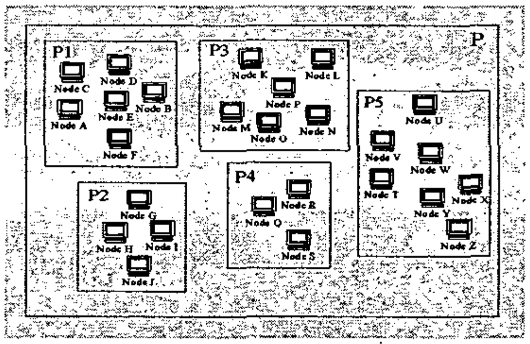

Fig. 4. A Two-layer Hierarchical System, P is the layer-1 circle, P1, P2, P3, P4 and P5 are layer-2 circles.

start	intervals	layer 1 successor	layer 2 successor
122	[122,123)	124: 001	143: 012
123	[123,125)	124: 001	143: 012
125	[125,129)	131: 112	143: 012
129	[129,137)	131: 112	143: 012
137	[137,153)	139: 022	143: 012
153	[153,185)	158: 012	158: 012
185	[185,249)	192: 001	212: 012
249	[249,121)	253: 012	253: 012

Fig. 5. Node 121: 012's fi nger tables in a two-layer Hierarchical P2P system with 3 landmark nodes, name space [0, 256)

finger table, in each entry, any peer can be the successor if it has the smallest nodeid in that interval. However, for lower layer finger table construction, we can only pick the peers within the same circle "012".

We also need to modify the structure of the file location table. Figure 6 shows the two-layer file location table for node 121: 012. The contents in the higher layer table are the same as Chord. It records the information of the peers which have copies of the corresponding files, no matter which lower layer circles these peers are in. However, for the lower layer table, it only records the information of the peers within the same layer-2 circle which store the corresponding files.

D. Routing and Retrieving Algorithm

In our system, routing and retrieving operations are done with a hierarchical scheme. For a m-layer system, this process takes 1 to m loops to finish. A great advantage of our scheme is that we use a current DHT algorithm as the underlying algorithm in each layer. However, in different layers, the finger table and file location table in that layer must be used. In our algorithm, when

Layer 1 File Location Table

Name	Fileid	Nodeid	NodeIP
File1	102	215: 210	X.X.X.X
File2	107	131: 112	X.X.X.X
File2	107	69: 012	X.X.X.X
File2	107	237: 020	X.X.X.X
⋮	⋮	⋮	⋮
FileK	121	214: 111	X.X.X.X

Layer 2 File Location Table

Name	Fileid	Nodeid	NodeIP
File2	107	69: 012	X.X.X.X
File3	118	143: 012	X.X.X.X
File3	118	212: 012	X.X.X.X
⋮	⋮	⋮	⋮
FileX	98	158: 012	X.X.X.X

Fig. 6. The Two-layer fi le location tables of node 121: 012

a client wants to retrieve a file, it generates a routing request with the required fileid as usual. However, it also includes its nodeid and circleid in the request. This routing request starts from the lowest circle where the client is in; Chord algorithm (with the lowest layer finger table) is used to find the peer whose nodeid is the numerically closest to the fileid. After receiving the request, this peer checks its lowest file location table, if a record of this file is found, it means one or more near-by peers in this circle have the copy, the identification information of these peers will be returned to the client. No further routing hops to other peers are needed and the routing request terminates immediately. Then, the client can retrieve the file from a near-by peer. If there's no record for that file exist in the lowest layer file location table, it means no peers within this lowest circle have a copy of the requested file, the routing request will be forwarded to the upper layer circle. The Chord routing algorithm and the upper layer finger table and file location table will be used to execute the routing request in that layer. As the routing request moves up, more and more peers will be included. At the last step, the highest layer finger table and file location table will be used, and all the system peers will be included. In any intermediate layer, if the location information of the requested file is found, the routing request will be terminated at that layer. The identification information of the peer which has the file and has the most number in common with the client's order information is returned to the client. The pseudo code of our routing and retrieving algorithm is shown in Figure 7.

Compare to Chord algorithm, our new algorithm has several advantages: First, it keeps the scalability property of the DHT based routing algorithm, a routing/retrieving operation definitely finishes within $O(logN)$ (N is the total number of peers in the system) steps; Second, it can greatly reduce the amount of wide-area network communications. The routing request does not have to reach the final destination in some cases, in any intermediate layer, it stops immediately after the location information of the requested file is found; Third, with the hierarchical architecture, we can easily find the closest peer which has the requested file. Thus, our system can greatly improve the routing and retrieval performance.

E. Information Maintenance

The benefits of our new algorithm come with extra cost. In Chord algorithm, when a new peer joins the system, only one global layer finger table and one file location table are generated. While in our system, we must execute the join operation several times with one for each layer to create multiple-layer tables. However, the extra cost is affordable: First, the storage space occupied by multi-layer tables is not a big issue since an entry in a table only occupies several bytes; Second, the total number of peers in the lower layer is smaller, the number of individual peers in the lower layer finger table is much less than in the higher layer. Also, the peers in lower layer circles are topologically closer to each other, the communication overhead to maintain the lower layer tables is smaller than in the higher layer; Third, the number of shared files in a lower layer circle is also much less than in higher layer. Thus, the maintenance overhead for these files is not very big.

In an m-layer system, for a shared file stored on a peer N, its location information is stored in m peers' file location tables simultaneously with one in each layer. The nodeids of these peers are the numerically closest to the fileid in the corresponding layers. When a new file is inserted, our routing algorithm can be used to find all the peers which will be used to store its location information. The process goes through from the lowest layer to the highest layer: In each layer, the final hop will arrive the peer whose nodeid is numerically closest to the fileid, thus, we can easily add the location information to the file location tables on these peers. Thus, one request is enough to modify the file location tables on m peers. Even in Chord, one request which will reach the final destination peer who is the numerically closest to the file is needed. Thus, we do not increase the overhead for file insert operation. For other operations such as update and delete, the situations are similar.

```
// n is the start clientid
// m is the hierarchy depth.
// key is the requested fileid
// c_id is the client's lowest circleid
find=FALSE;
n.routing_retrieving(key)
  layer=m;
  n'=n;
  while(layer!=0)
   {
    n'=n'.layer_routing_retrieving(layer.c_id.key);
    if(!find)
        layer=layer-1;
    else
        break;
   }
 return n';
n'.layer_routing_retrieving(layer.c_id.key)
  current_finger_table=finger_table[layer];
  n'.underlying_routing_algorithm(current_finger_table.key);
  if(key is in n' file location table)
                  // one or more copies are found
   {
     compare c_id with the client;
     n'= the node has the most number in common;
     find=TRUE;
   }
  return n';
```

Fig. 7. Our Routing and Retrieving Algorithm

F. Effects of landmark nodes and hierarchy depth

The different number of landmark nodes will affect system performance. As more landmark nodes are used, more digits are needed to represent the order information for each peer. More lower layer P2P circles will be created, and the average number of peers in each circle will reduce. At the same time, the average link latency between any two peers in the lowest layer circles will decrease compare to the latency in the lowest layer circles generated by using less number of landmark nodes. On the other hand, too many landmark nodes will increase the overhead for node join operations. More "ping" messages will be sent. Thus, finding a suitable number of landmark nodes is important. We conducted simulation experiments to evaluate the effects of different number of landmark nodes.

Hierarchy depth also affects the system performance. As more layers are introduced, more finger tables and file location tables are needed, which increases the system maintenance overhead. On the other hand, more routing and retrieving information can help us to find a nearby copy more quickly. A tradeoff is necessary to achieve the satisfactory performance with affordable maintenance overhead. In our simulations, we also evaluate the effects of the hierarchy depth.

V. Performance Evaluation

In this section, we present simulation results demonstrating the benefits of our design. First, we describe the simulation environment, including the network model to be used in our experiments, the workload generation mechanism and the performance metric used to evaluate the performance. Following this, we present the evaluation and analysis of the experimental results.

A. Simulation Environment

We choose GT-ITM Transit-Stub (TS model) as the primary network topology model for our simulations. TS model is an internetwork topology model proposed by E. Zegura in [15]. A TS network is composed of interconnected transit and stub domains. Transit domains function more like Internet backbone while stub domains work more like local area networks. Thus, it can reflect the hierarchical character of the internetwork and it is more accurate than a random network model. In our emulated networks, the total number of nodes varies from 1000 to 10000. The number of nodes in each stub domain varies from 16 to 20 according to the different total number of nodes. We use SRS to represent the simple retrieving solution, TRR to represent our topologically-aware routing and retrieving algorithm. In our simulation experiments, unless specified, we use a two-layer hierarchy system with four landmark nodes used to generate circleids for lower layer P2P circles. 100000 randomly generated pseudo file requests are used as the system workload.

In the real world, files have different popularity. A large percentage of shared files are seldom accessed and only have a small number of copies in the system. On the other hand, a small set of hot files are requested more frequently than other files and have multiple copies stored on different peers. To reflect this fact, we use the following file distribution: "10-30-60". Here, we define 10% of the shared files have 10 copies stored on different peers, 30% of the files have 5 copies, and the rest 60% of the shared files only have one copy. The location of each copy is chosen randomly.

We use the average routing latency for one request as the performance metric to evaluate the routing performance. We use the link latency between the client and the peer from which to retrieve the requested file as the metric to evaluate the retrieval performance. For retrieving files, for any files which have multiple copies on different peers, Chord algorithm is unable to determine the location of the closest copy, we assume it will randomly choose one from these peers. While in SRS and TRR algorithms, the peer which has the most number of digits of order information in common with the client is chosen.

B. Routing Performance

In the first set of simulation experiments, we evaluate the routing performance. Figure 8 shows the result. For all different sized networks, TRR achieves the best performance. The average routing latency in TRR is only about 43.9% to 51.2% of Chord. For SRS, it can only reduce the retrieving overhead

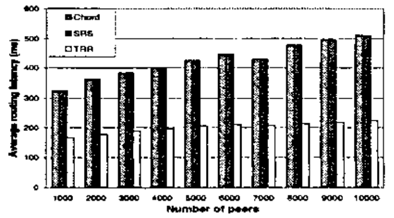

Fig. 8. The routing performance comparison

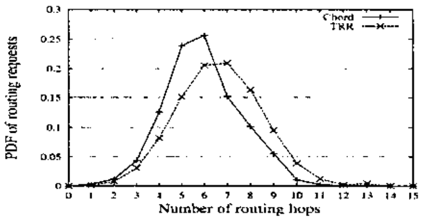

Fig. 9. The routing hops distribution in a 10000-node network

and can not reduce the routing overhead. SRS still uses the same routing algorithm in Chord, thus, they have the same routing performance. In general, as the total number of peers increases, the average routing latency also increases since more routing hops are needed to search a file. There's one exception, the routing overhead in a 7000-node network is smaller than in 6000-node network, however, this anomaly is caused by the different parameters we chosen to generate TS networks in these two sizes.

In our experiments, the file requests are randomly generated. Thus, the access rate for each file is the same. However, in real world, popular files which have more copies on different peers are also have higher request rates than other files [16]. We can expect TRR to achieve better performance since for hot files with many copies, the possibility of finding a copy on near-by peers is much higher than other files.

To better understand the benefits of using a hierarchical routing scheme in TRR, we analyze the routing cost distribution in different algorithms. Figure 9 shows the probability density distribution (PDF) curves for Chord and TRR. The data is collected on a 10000-node TS network. For randomly 100000 routing requests, in TRR, the percentages of the requests finished within a small number of hops [0, 6] are higher than in Chord, while in Chord, the percentages of the requests which need more hops [7,14] to finish are higher than in TRR. This is because, in Chord, all the requests will reach the destination (the peer who is responsible for the requested key), while in TRR, the requests can finish at different layers. In Chord, the average number of routing hops for each request is 6.4933, while in TRR, the value is only 5.0108.

Table I shows the percentage of routing requests finished in the lower layer circles in TRR. As the total number of peers increases, the percentage also increases. This is because we use 4 landmarks nodes for different network sizes. In a large network, the average number of peers in the lower layer circle is greater than a small network. TRR have better chance to find the location information of the requested files in the lower layer.

In TRR algorithm, the routing performance improvement comes from two aspects: except the routing requests finished in the lower layer circles, the requests finished in the higher layer also get the benefits from a hierarchical architecture. Table II shows in TRR, for those routing requests which can not finish in

TABLE I

REQUESTS FINISHED IN THE LOWER LAYER IN TRR

Node num	2000	4000	6000	8000	10000
Percent(%)	3.18	4.59	8.67	12.74	19.16

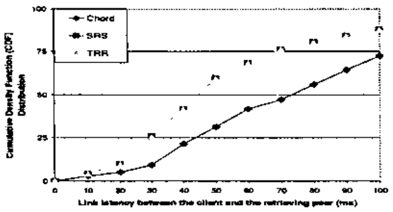

Fig. 10. The Cumulative Distribution Function (CDF) of the link latency between the client and the peer to retrieve fi le

the lower layer, the average number of hops for each request and the number of hops taken in the lower layer circles. Since the average link latency between any two peers in the lower layer circles is much smaller than the higher layer, TRR replace a large percentage of hops in the higher layer with the hops in the lower layer, it also reduce the routing overhead for those routing requests.

TABLE II

ROUTING HOPS PER REQUEST DISTRIBUTION IN TRR

Node num	2000	4000	6000	8000	10000
total	5.45	5.94	6.26	6.43	6.59
layer-2	3.76	4.15	4.52	4.89	5.12

C. Retrieval Performance

The ultimate goal of routing procedures is to find the peers for the retrieving operations. Thus, retrieval performance is also very important. In this experiment, we evaluate the retrieval performance of different algorithms. Figure 10 shows the link latency distribution of the client and the peer which was chosen for the retrieving operation. Chord does not consider the topological information, thus, it can only randomly choose a peer from the peers which have the requested file. For SRS and TRR, both of them can find the closest peer to the client which has the longest number of digits in common on the order information with the client. As shown in Figure 10, in SRS and TRR algorithms, 60% of the peers chosen for the retrieve operations have link latency less or equal to 50ms to the client, while in Chord, the number is only 31%. 89% of peers chosen in SRS and TRR have link latency within 100ms to the client, however, only 72% in Chord. Clearly, our algorithms have better retrieval performance.

D. Effects of landmark nodes

We also evaluate the effects of the different number of landmark nodes. The result is shown in Figure 11. Since Chord and SRS algorithms do not use landmark nodes for their routing procedures, the different number of landmark nodes does not affect the routing performance. While in TRR algorithm, as more landmark nodes are used, more digits are needed to represent a peer's order information. For any two peers, to be grouped into the same layer-2 circle, they must have more number of digits in common in their order information. Thus, more layer-2 P2P circles will be created, the average number of peers in each circle will reduce. For TRR algorithm, the average routing hops in

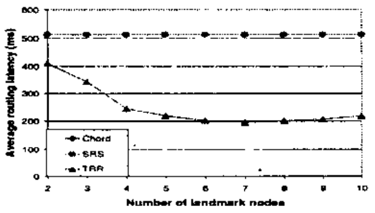

Fig. 11. The Effects of different number of landmark nodes, 10000-node network

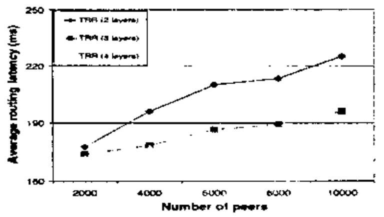

Fig. 12. The Effects of hierarchy depth

the lower layer per request will decrease. However, the average latency per hop also reduces because the peers in those smaller lower layer circles are much closer.

The decreased number of hops per request in the lower layer means the increased number of hops per request in the higher layer. Since the average latency per hop in the higher layer is greater than in lower layer. thus the average routing latency tends to grow as the number of landmark nodes increase. On the other hand, the average latency of the hops in the lower layer decreases as the number of landmark nodes increases, the overall routing latency should decrease. The actual routing latency is determined by these two factors. As shown in figure 11, as the number changes from 2 to 7, the system can achieve significant performance improvement. In this case. the decreased average latency per hop in the lower layer dominates. For the number changes from 8 to 12, the average routing latency will increase. Here, the first factor takes major effect. In this experiment, 7 landmark nodes are the best choice for this 10000-node network.

E. Effects of hierarchy depth

We also conduct experiments to evaluate the effects of the hierarchy depth. We test the routing performance for 2, 3 and 4 layers and the result is shown in Figure 12. Clearly, hierarchy depth also affects the system routing performance. As the number of layers increases, the routing performance is improved. However, for small systems with 5000 peers, the difference is trivial. For the system with more peers, when the number of layers changes from 2 to 3, system can achieve considerable performance improvement. However, the improvement is trivial when we change the number from 3 to 4. As the number of layers increases, more maintenance work has to be done for more finger table and file location tables. Thus, choosing a suitable number of layers is also important to achieve the optimal performance. For our experiments, 3 layers are good enough.

VI. RELATED WORKS

Chord [5] [17], Pastry [6], Tapestry [7] and CAN [8] are DHT based routing algorithms. In these systems, an elegant and efficient routing data structure is generated. However, the negligence of the peers' topological information seriously hurts the routing and retrieval performance. Although DHT algorithms can achieve the optimal routing performance in terms of routing hops, they can not achieve the minimal routing latency. Also, in DHT systems, the retrieving problem is not well considered.

Ratnasamy and Shenker [14] pointed out that P2P or other large-scale network applications could potentially benefit from some level of knowledge about the relative proximity between its participating nodes. They suggested allocating the topologically adjacent peers with congruent identifiers together, and they applied this idea on CAN with significant improvement on the routing performance. HIERAS [18] and Coral [19] also use the topological characteristics and create hierarchical architecture to boost the routing performance for standard DHT routing algorithms. There's two differences compare to our system: First, in both algorithms, a routing request will terminate only when it reaches the final destination peer which is responsible for the requested fileid, while in our system, the routing operation can terminates in any layers, thus our algorithm can effectively reduce the amount of wide-area network communications. Second, none of the algorithms addressed the retrieving problem. Both algorithms can not guarantee to find out the closest copy for the client.

In Kazaa [3], superpeers are used to improve the routing performance in unstructured P2P applications. SBARC [20] and Brocade [21] also use superpeers to improve the routing performance in DHT based systems. A superpeer is a peer which has more computer resources (such as CPU speed, network bandwidth and storage size, etc.) than an ordinary peer. In these systems, the routing tasks are mostly taken by superpeers. However, the failure of a superpeer will result in serious performance downgrade. Also, the retrieving problems are not fixed in superpeer based systems.

VII. Conclusion and Future Work

We started with the discussion of the routing/retrieving problems in current DHT systems followed by our new algorithms to address these problems. Our system takes the peers' topological information into account for efficient routing/retrieving operations. We use distributed binning scheme to determine a peer's position (topological information) in a k-dimensional space. In the simple solution, we use this information to help the client choose the suitable peer for file retrieving. In TRR algorithm, we further utilize this information to reduce the routing overhead. We generate multiple P2P circles in different layers. In any intermediate layer, the routing procedure could be finished as soon as the location information of the requested file is found. Our simulation results prove that our strategy can achieve significant routing and retrieval performance improvement over the current DHT algorithms. For the future work, more simulation experiments will be conducted. We plan to use larger scale network (more peers) to evaluate the performance. Different network models such as Inet [22] and BRITE [23] will also be used. We will try to generate a formula to determine the most suitable numbers of the landmark nodes and the hierarchy depth for a given set of peers.

References

[1] Napster, "http://www.napster.com."

[2] Gnutella, "http://www.gnutella.wego.com."

[3] KaZaA, "http://www.kazaa.com/."

[4] BitTorrent, "http://www.bittorrent.com/."

[5] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications." Technical Report TR-819, MIT., Mar. 2001.

[6] A. I. T. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," in *Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Heidelberg, Germany*, pp. 329–350, Nov. 2001.

[7] B. Zhao, J. Kubiatowicz, and A. Joseph, "Tapestry: An infrastructure for fault-tolerant widearea location and routing." Technical Report UCB/CSD-01-1141, U.C.Berkeley, CA, 2001.

[8] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content addressable network." Technical Report, TR-00-010, U.C.Berkeley, CA, 2000.

[9] J. Kubiatowicz, D. Bindel, P. Eaton, Y. Chen, D. Geels, R. Gummadi, S. Rhea, W. Weimer, C. Wells, H. Weatherspoon, and B. Zhao, "OceanStore: An architecture for global-scale persistent storage," in *Proceedings of the 9th international Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), Cambridge, MA*, pp. 190–201, Nov. 2000.

[10] P. Druschel and A. Rowstron, "Past: A large-scale, persistent peer-to-peer storage utility," in *the 8th IEEE Workshop on Hot Topics in Operating Systems (HotOS), Schoss Elmau, Germany*, May 2001.

[11] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Wide-Area cooperative storage with CFS," in *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP), Banff, Alberta, Canada*, pp. 202–215, Oct. 2001.

[12] D. R. Karger, E. Lehman, F. T. Leighton, R. Panigrahy, M. S. Levine, and D. Lewin, "Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web," in *Proceedings of the 29th annual ACM symposium on Theory of computing ACM Symposium on Theory of Computing (STOC), El Paso, TX*, pp. 654–663, May. 1997.

[13] N. I. of Standards and Technology, "http://csrc.nist.gov/publications/fi ps/fi ps180-2/fi ps180-2.pdf."

[14] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, "Topologically-aware overlay construction and server selection," in *Proceedings of IEEE INFOCOM'02, New York, NY*, Jun. 2002.

[15] E. W. Zegura, K. L. Calvert, and S. Bhattacharjee, "How to model an internetwork," in *Proceedings of the IEEE Conference on Computer Communication, San Francisco, CA*, pp. 594–602, Mar. 1996.

[16] S. Saroiu, P. K. Gummadi, and S. D. Gribble, "A measurement study of peer-to-peer fi le sharing systems," in *Proceedings of Multimedia Computing and Networking (MMCN), San Jones, CA*, Jan. 2002.

[17] F. Dabek, E. Brunskill, M. F. Kaashoek, D. Karger, R. Morris, I. Stoica, and H. Balakrishnan, "Building peer-to-peer systems with chord, a distributed lookup service," in *the 8th IEEE Workshop on Hot Topics in Operating Systems (HotOS), Schoss Elmau, Germany*, pp. 195–206, May 2001.

[18] Z. Xu, R. Min, and Y. Hu, "HIERAS: A DHT-Based Hierarchical Peer-to-Peer Routing Algorithm," in *the Proceedings of the 2003 International Conference on Parallel Processing (ICPP'03)*, (Kaohsiung, Taiwan, ROC), October 2003.

[19] M. J. Freedman and D. Mazieres, "Sloopy Hashing and Self-Organized Clusters," in *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS), Berkeley, CA*, Feb 2003.

[20] Z. Xu and Y. Hu, "SBARC: A Supernode Based Peer-to-Peer File Sharing System," in *proceedings of the 8th IEEE Symposium on Computers and Communications (ISCC'03)*, (Kemer-Antalya, Turkey), June 2003.

[21] B. Zhao, Y. Duan, L. Huang, A. Joseph, and J. Kubiatowicz, "Brocade:landmark routing on overlay networks," in *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS), Cambridge, MA*, March 2002.

[22] C. Jin, Q. Chen, and S. Jamin, "Inet: Internet topology generator." Report CSE-TR443-00, Department of EECS, University of Michigan, 2000.

[23] A. Medina, A. Lakhina, I. Matta, and J. Byers, "Brite: An approach to universal topology generation," in *Proceedings of the International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunications Systems (MASCOTS'01), Cincinnati, OH*, Aug. 2001.