Available online at www.sciencedirect.com
ScienceDirect
Future Generation Computer Systems 23 (2007) 201–208
FGCS FUTURE GENERATION COMPUTER SYSTEMS
www.elsevier.com/locate/fgcs

A structured hierarchical P2P model based on a rigorous binary tree code algorithm

Hongjun Liu[a,*], Ping Luo[a], Zhifeng Zeng[b]

[a] Department of Computer Science, Tsinghua University, Beijing 100084, China
[b] Department of Computer Science and Technology, Yunnan Police Officer Academy, Yunnan KunMing 650223, China

Received 15 October 2005; received in revised form 26 April 2006; accepted 27 April 2006
Available online 23 June 2006

Abstract

One of the main problems in peer-to-peer systems is how to balance the lookup latency, the number of transmitted messages and the network expansibility. In this paper, we present a P2P model based on a rigorous binary tree code algorithm. Our model is robust and efficient with constant lookup latency and constant message transmission. More importantly, the model has good expansibility and is suitable for super-scale networks.

Keywords: P2P; Super-peer; Rigorous binary tree code algorithm; DHT

1. Introduction

Research of the peer-to-peer (P2P) [10,15] technique focuses on the sharing of computer resources (including data files, storage space, and CPU processing power) among users, without queries passing through the intervention and service of intermediary servers. In this paper, we only discuss file-sharing P2P systems.

The basic models of file-sharing P2P systems include three styles:

(1) Totally distributed P2P models such as those of Gnutella [2] and Freenet [7]. In this type of model, each peer only knows the information of its logic neighbors. When a peer wants to inquire about a resource, it sends a query message to all of its neighbors. Its neighbor will answer the query if it contains a file related to the query, or forward the query to all of its own neighbors. The process will continue until the TTL [14] is over or the needed resource is discovered. Each peer's function and role are equal, so any peer's joining or leaving does not affect the system much. As a result, it is robust in fighting against attack. But the system is flooded when all query messages are forwarded, often resulting in broadcast storms and wasted network bandwidth.

(2) Fully centralized P2P models, such as Napster [1]. In this type of model, there is a central server. The server stores all the indices of the shared resources and offers a query service for its users. By querying the central server, each peer can get its needed index. For any query, it takes only two messages and a two-hop latency to find a needed index. Except for the server, any peer's joining or leaving affects the system little. System management is convenient. But all queries must pass through the server, so the server becomes a potential bottleneck of the system. Once the server has a problem, all querying operations will be affected. If a hacker breaks into the server, the overall system will be unsafe. In addition, the processing power of a server is limited, so this type of model has bad expansibility.

(3) Structured P2P models, i.e., the DHT-based P2P model (DHT [4,5], distributed hashing table). In this type of model, such as Chord [6], every peer is assigned a unique figure identity marked with Peer-ID, and organized by a certain data structure. Each peer maintains several peers' Peer-ID information for route querying. According to some hash function algorithms (such as MD5 [18] and SHA-1 [20]), every shared resource is mapped to have a unique key value marked with Key-ID in the system. The shared resource is stored in a peer by a uniform rule. Hence, through associating a Key-ID with a Peer-ID by a certain rule, a query for a resource is transformed to be a lookup for a peer who has been ranked well in an order. Though this model has decreased the cost of

* Corresponding author.
E-mail addresses: liuhongjun666@126.com (H. Liu), luop@mail.tsinghua.edu.cn (P. Luo), hhzzf@163.com (Z. Zeng).

0167-739X/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2006.04.016

searching in a logarithmic layer in proportion to the number of the peers in a system, for example, in the Chord [6] system, the lookup latency may exceed 20 hops in a large-scale network with more than 1000,000 peers. This is still oppressive.

From the above analysis we can see that, when P2P technology is applied on a large-scale network, the problem of how to balance the lookup latency, the number of transmitted messages and the scalability of the network must be addressed.

In this paper, we present a P2P design that employs the super-peer concept, and explore the balance of lookup latency, transmitted messages and network expansibility.

There are many studies that refer to the concept of super-peers, such as in [3,8,9,13,21,22,24,26,27]. The concept is not discussed in this paper.

Our design offers constant lookup latency and a constant number of transmitted messages. More importantly, our design can automatically be customized for different scales of networks, thereby eliminating bottlenecks, by adopting the rigorous binary tree code algorithm.

The remainder of this paper is organized as follows: Section 2 describes the rigorous binary tree code algorithm and mapping theorem which our design uses. Section 3 presents the theory behind our design. Section 4 presents our design. Section 5 gives related works. Section 6 gives our conclusion.

2. Rigorous binary tree code algorithm and mapping theorem

To give readers a better understanding of our model, we first provide the definition of a rigorous binary tree: for a random node of a binary tree, if it has at least one child node, its left child node and right child node must exist at the same time. If this condition is satisfied, the binary tree is defined as a rigorous binary tree.

Rigorous binary tree extension: after a random leaf of a rigorous binary tree produces two children nodes, the original rigorous binary tree becomes a new rigorous binary tree. This is called rigorous binary tree extension.

Rigorous binary tree code algorithm: the letter "T" represents a rigorous binary tree, "A" represents a random node in T, "h_a" represents the depth of node A, and "num_a" represents the code of node A. The code of T's root node was set as 0. According to the following rule, we can calculate the code of A's children. The code of A's left child is equal to num_a, and its depth is $h_a + 1$. The code of A's right child is equal to $(\text{num}_a + 2^{h_a})$, and its depth is the same as the depth of A's left child.

Fig. 2.1 illustrates a rigorous binary tree. When we extend its leaf node G in Fig. 2.1(a) by adding two children nodes to G, the rigorous binary tree becomes that described in Fig. 2.1(b). According to the rigorous binary tree code algorithm, node A is the root node, so its code and depth are (0, 0). Node B is A's left child, so B's code is equal to A's code (0) and B's depth is A's depth plus one ($0 + 1 = 1$). Node C is A's right child, so C's code is equal to ($0 + 2^0 = 1$) and C's depth is equal to B's depth (1). In the same way, we can compute the remaining nodes' code and depth as described in Fig. 2.1(c).

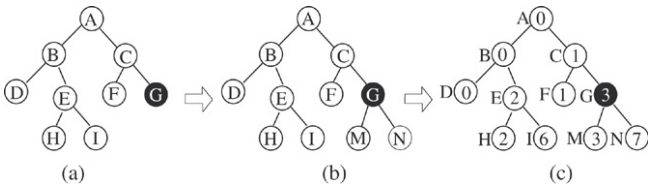

Fig. 2.1. The extending and coding of a rigorous binary tree.

Rigorous binary tree mapping theorem: for any one integer N ($N >= 0$), there is one and only one leaf node X whose code (num_x) and depth (h_x) can accord with $\text{num}_x = N\%2^{h_x}$, among all leaf nodes in a fixed rigorous binary tree. (Here % denotes modular arithmetic.) For the sake of brevity, the proof process is not included in this report.

3. Design principles

In our model, all peers in an overlay are organized into several groups by a uniform mapping rule. In each group, there is at least one super-peer. All super-peers form an upper overlay named the super-peer layer. Every super-peer has a code value and a depth value calculated with the rigorous binary tree code algorithm. Fig. 3.1 illustrates the basic architecture of our system. When a super-peer reaches its limit, it will select a high-powered peer as a new super-peer who will be assigned a code/depth pair computed with the rigorous binary tree code algorithm, and the original super-peer will share approximately half of its load with the new super-peer according to the uniform mapping rule. When a peer (for example, P_{12} in Fig. 3.1) wants to inquire about a file with a given key, first of all, it queries (message m_1 in Fig. 3.1) its local super-peer (SP_2) with the given key. Then, the local super-peer (SP_2) will calculate the given key with the uniform mapping rule. If the calculated result points to the super-peer (SP_2) itself, the super-peer (SP_2) will find a peer who is responsible for the given key from its registered information. Otherwise, the query message (m_2) will be transmitted to another super-peer (SP_3) pointed to by the calculated result, and the super-peer (SP_3) will process the query message finally (m_3).

With these mechanisms, our model has good scalability and offers constant lookup latency in terms of hop count and a constant number of transmitted messages.

4. Design model

Our design model includes:
(1) How to initialize the system.
(2) How a peer joins the system.
(3) How to store or request a file with a key.
(4) How to produce a new super-peer.
(5) How to manage the failure of peers and super-peers.
(6) How to adapt to a super-scale network.

4.1. The initialization of a system

We suppose that the initiating peer of the system is peer C, and we designate its address as IP_c. The initial peer acts as

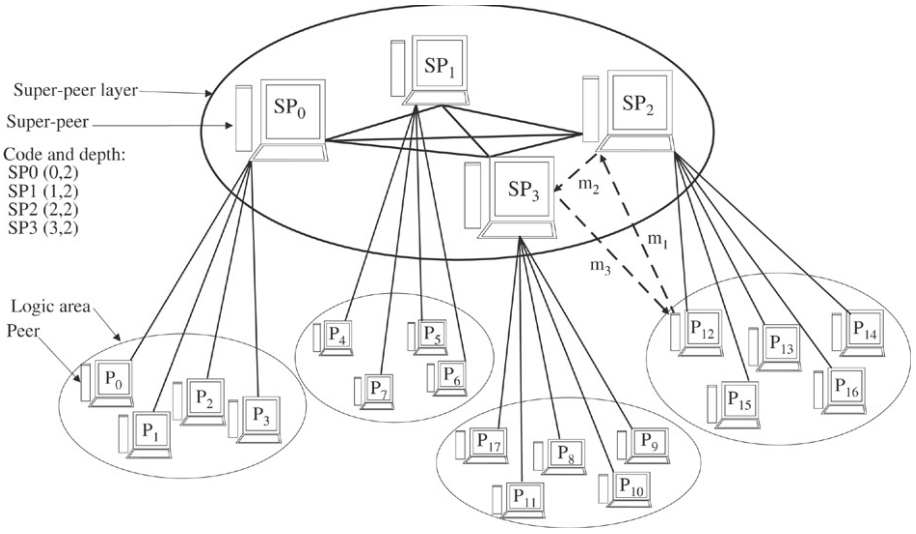

Fig. 3.1. The architecture of peers and super-peers.

Table 4.1
Super-peer table

Name	IP address	Code (num)	Depth (h)
C	IP_c	0	0

Table 4.2
Registered-peer table of super-peer C

$IP_x\%2^{(h_c+1)} = num_c$			$IP_x\%2^{(h_c+1)} = num_c + 2^{h_c}$		
Name	IP address	Key value	Name	IP address	Key value

the first super-peer and, according to the rigorous binary tree code algorithm, holds the root node's code and depth ($num_c = 0, h_c = 0$). At the same time, it produces two tables: the super-peer table and registered-peer table.

The super-peer table is used for noting the correlated information of all super-peers in the system and held by every super-peer. The contents in the super-peer table include the name, IP address, code value and depth value of each super-peer. Table 4.1 illustrates the contents of the super-peer table.

The registered-peer table is used for noting the correlated information of all peers registered in a super-peer. Its contents contain the peer's name and IP address, and the key value which a file is mapped to with a uniform mapping rule. For the convenience of super-peer extension, all registered peers are divided into two parts: one according to $IP_x\%2^{(h_y+1)} = num_y$ and the other according to $IP_x\%2^{(h_y+1)} = num_y+2^{h_y}$, in super-peer Y's registered-peer table. (Here Y denotes any super-peer and X denotes any registered peer in Y's registered-peer table.) Table 4.2 illustrates the contents of super-peer C's registered-peer table.

4.2. Process for peers to join the system

For any peer to join the system, it must know at least one peer who has been registered in a super-peer. Through the registered peer, the peer can know a super-peer in the system. In this paper, we did not describe how to know a registered peer.

We suppose that peer B has known super-peer A in the system, and B wants to join the system. The steps of the process are as follows:

(1) B sends a message with its IP address (IP_b) to super-peer A.

(2) After receiving the message, A fetches a super-peer's code and depth from the super-peer table.

(3) A calculates $IP_b\%2^{h_x}$, and compares the result with num_x. (Here X denotes the fetched super-peer, num_x denotes X's code, h_x denotes X's depth and % denotes modular arithmetic.)

(4) If $(IP_b\%2^{h_x}) \neq num_x$, then A fetches the next super-peer's code and depth. Return to step (3).

(5) If $num_x = num_a$, i.e., X is super-peer A, then A accepts B's application, adds B's information to its registered-peer table and notifies B.

(6) If $num_x \neq num_a$, then A forwards the message to super-peer X. X will accept B's application, add B's information to its own registered-peer table and notify B.

To explain how a peer joins the system, we give the following example.

Suppose that there are five super-peers A, B, C, D and E, whose codes and depths are A(0,1), B(1,3), C(5,4), D(13,4) and E(3,2) respectively, as illustrated in Fig. 4.1. Suppose that there is a peer X whose IP address is $IP_x = 27$, and X wants to join the system. X randomly selects a super-peer it knows and sends a message with its IP_x (27) to the super-peer. (Here we assume it is super-peer A.) A takes its own code and depth (0, 1) and calculates $27\%2^1 = 1$. The result 1 is not equal to its num_a (0), so it continues by taking B's code and depth (1, 3), and calculates $27\%2^3 = 3$. The result 3 is not equal to B's $num_b(1)$, so A proceeds to take the codes and depths of C, D and E in turn. After the calculations, super-peer A knows that E's code and depth (3, 2) can match $3 = 27\%2^2$, and forwards X's message to E. Finally, super-peer E accepts X as its registered peer and notifies X.

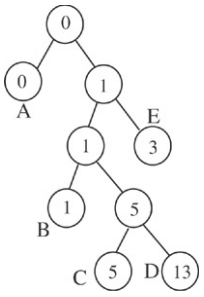

Fig. 4.1. Peer's joining.

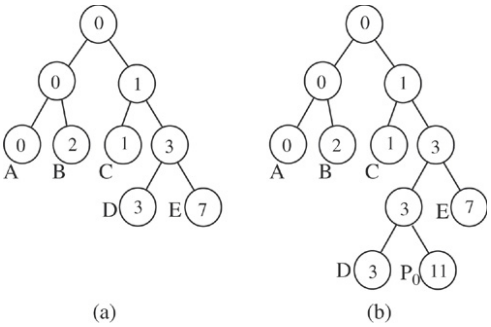

Fig. 4.2. Super-peer's extending.

From this process we can know that a peer's joining affects no other peers except for one or two super-peers, and it brings three messages at most.

4.3. Storing (or requesting) a document with a key

When a peer wants to store a file with a key "ID" in the system, first, it sends a message with the key "ID" to its local super-peer. Then, the local super-peer finds a super-peer X whose code (num_x) and depth (h_x) accord with $\text{num}_x = \text{ID}\%2^{h_x}$, using a process similar to the one described for the joining of a peer.

If X is the local super-peer itself, then the local super-peer calculates $\text{ID}\%2^{(h_x+1)}$. (1) If the result is num_x, then the local super-peer fetches a peer Y from the left part of its registered-peer table. (2) If the result is $\text{num}_x + 2^{h_x}$, then the local super-peer fetches a peer Y from the right part of its registered-peer table. The local super-peer then fills the key "ID" in the item of peer Y's key value and sends Y's address to the initial peer.

Otherwise, the local super-peer forwards the message with the key "ID" to the super-peer X. Then X performs the same action, in its registered-peer table, fetching a matching peer Y and sending Y's address to the initial peer, as described above.

Finally, the initial peer contacts Y and sends the file with the key "ID" to Y for storage.

The process of requesting a file with a key "ID" is similar to that for storing a file with a key "ID".

From this process we can know that a query produces at most three messages and three hops' latency. (The first message and latency are from the initial peer to the local super-peer, the second are from the local super-peer to the remote super-peer, and the third are from the remote super-peer to the initial peer.)

4.4. Producing a new super-peer

When the number of peers registered in a super-peer reaches the quantity limit, in order to balance the load and avoid a bottleneck, the super-peer will select a high-powered peer from its registered-peer table as a new super-peer, code it with the rigorous binary tree code algorithm and share one part of its load with the new super-peer.

For clarity, we provide an example.

Suppose that there are 5 super-peers A, B, C, D, and E, with codes and depths of A(0,2), B(2,2), C(1,2), D(3,3), E(7,3) respectively, as illustrated in Fig. 4.2(a), and D has 12 registered peers. The super-peer table and D's registered-peer table are as described as in Tables 4.3 and 4.4. (∗ *For calculation convenience, we used small integers to express the peer's IP address and key value.*)

Table 4.3
Super-peer table

Name	IP address	Code (num)	Depth (h)
A	IP_a	0	$h_a = 2$
B	IP_b	2	$h_b = 2$
C	IP_c	1	$h_c = 2$
D	IP_d	3	$h_d = 3$
E	IP_e	7	$h_e = 3$

Table 4.4
Registered-peer table of super-peer D

$(\text{IP}_x\%2^{h_d+1} = \text{num}_d)\text{IP}_x\%2^4 = 3$			$(\text{IP}_x\%2^{h_d+1} = \text{num}_d + 2^{h_d})\text{IP}_x\%2^4 = 11$		
Name	IP address	Key value	Name	IP address	Key value
P_0	19		P_6	43	
P_1	35		P_7	139	1035
P_2	67	1027	P_8	171	
P_3	83		P_9	219	1115
P_4	163	1155	P_{10}	235	
P_5	243	1331	P_{11}	251	1259

Now D wants to produce a new super-peer. The process is as follows:

(1) From its registered-peer table, D selects a high-powered (better processing power, more bandwidth, greater storage space, longer on-line time) peer as a new super-peer. E.g., P_0 is selected.

(2) According to the rigorous binary tree code algorithm, D recalculates the code and depth of D and P_0. The final result is $\text{num}_d = 3$, $\text{num}_{p_0} = 11$, $h_d = 4$, and $h_{p_0} = 4$, as illustrated in Fig. 4.2(b).

(3) D updates the super-peer table, and sends the update message to every super-peer (A, B, C, and E). The updated super-peer table is described in Table 4.5.

(4) D sends the super-peer table and the right part of the peers' information in its registered-peer table to the new super-peer P_0.

(5) D sends a message that the super-peer has become P_0 to each peer whose information has been sent to P_0 in the right

Table 4.5
Updated super-peer table

Name	IP address	Code (num)	Depth (h)
A	IP_a	0	$h_a = 2$
B	IP_b	2	$h_b = 2$
C	IP_c	1	$h_c = 2$
D	IP_d	3	$h_d = 4$
P_0	IP_{p0}	11	$h_{p0} = 4$
E	IP_e	7	$h_e = 3$

Table 4.6
Updated registered-peer table of super-peer D

$IP_x\%2^5 = 3$			$IP_x\%2^5 = 19$		
Name	IP address	Key value	Name	IP address	Key value
P_1	35		P_3	83	
P_2	67	1027	P_5	243	1331
P_4	163	1155			

Table 4.7
Registered-peer table of super-peer P_0

$IP_x\%2^5 = 11$			$IP_x\%2^5 = 27$		
Name	IP address	Key value	Name	IP address	Key value
P_6	43		P_9	219	1115
P_7	139	1035	P_{11}	251	(~~1259~~)
P_8	171	(1259)			
P_{10}	235				

part of D's registered-peer table. In our example, those peers include P_6, P_7, P_8, P_9, P_{10} and P_{11}.

(6) D and P_0 update their registered-peer tables respectively. Super-peer D's new registered-peer table is described in Table 4.6, and P_0's is in Table 4.7. When a super-peer X updates its registered-peer table, it should check whether the key value (ID) stored in a peer Y can accord with Y's IP address. For example, in Table 4.7, the key value (1259) stored in P_{11} does not accord with P_{11}'s IP address (251), because $1259\%2^5$ is not equal to $251\%2^5$. Hence, super-peer P_0 should find another suitable peer to store the file whose key value is 1259, e.g., select P_8. Then P_0 lets P_{11} transmit the file with the key value (1259) to P_8.

4.5. Managing the failure of peers and super-peers

A peer's failure can make a file/key pair unavailable if the file/key pair has been stored in the peer.

To avoid this case, we adopted a redundancy mechanism; i.e., when a super-peer wants to fetch a peer from its registered-peer table to store a file/key pair, it fetches two peers instead of one. Thus, if a peer fails, the file/key stored in that peer is still available from the other one.

To manage two peers storing the same file/key pair, we can make them contact each other periodically. If one peer detects that the other is unreachable over a limited time period, the former can message its local super-peer. Then, the local super-peer can select another peer instead of the failed one to store the same file/key pair.

A super-peer's failure can affect a subset of peers in a logic area.

To resolve this problem, in a similar fashion, we adopted a redundancy mechanism. When any one super-peer wants to produce a new super-peer, it selects k high-powered peers as a new group of super-peers with the same contents. Commonly, we select $k = 2$–3. In Fig. 4.3, we describe the architecture with redundancy.

Thus, if the number of the failed super-peers in any group is less than k, the system can still work well. Furthermore, by selecting some new peers as super-peers, the living super-peers can recover the system to a normal level with k super-peers in each group.

In one way, the redundancy mechanism increases the size of the super-peer table, but on the other hand, it decreases the average load of each super-peer and makes the system more reliable.

4.6. Adapting to a super-scale network

For a super-scale network with about 10^9 peers, if a super-peer can serve about 1000 peers excluding super-peers, there will be about 10^6 super-peers. Hence, in a single super-peer layer system, besides peers, each super-peer has to serve about 10^6 super-peers. This will be very difficult for a super-peer. We address how to solve this problem in our system, by adopting a multi-layer query mechanism.

For example, we can set a super-peer's parameters as follows. For any one super-peer A, if its depth reaches 10, then it produces a sub-super-peer table whose form is the same as the super-peer table, and enters its IP address and name in the table acting as a default super-peer. When A reaches its limit, it selects two peers (suppose that they are peer B and peer C) as super-peers who are coded as A's left child and right child respectively, and adds the information of B and C into the sub-super-peer table, instead of updating its super-peer table. Then, super-peer A only sends the sub-super-peer table to B and C, and distributes its whole load to B and C according to the uniform mapping rule. Henceforth, super-peer A will only serve those super-peers who are in its super-peer table or in its sub-super-peer table. When a super-peer in a sub-super-peer table wants to produce a new super-peer and updates the sub-super-peer table, it will only message those super-peers in its sub-super-peer table, in addition to the new super-peer. When a super-peer in a sub-super-peer table wants to query a file/key pair, if it cannot find a super-peer that matches the key, it forwards the query to the default super-peer in its sub-super-peer table.

With this approach, for a super-scale network with about 10^9 peers, there will be two super-peer layers, as illustrated in Fig. 4.4. Layer 1 consists of those super-peers who are in the super-peer table. Layer 2 consists of those who are in the sub-super-peer table. Each super-peer in layer 1 serves the super-peers in its sub-super-peer table and those in layer 1. Each super-peer X in layer 2 serves its registered peers and those super-peers who are in the same sub-super-peer table with X. Via layer 1 and layer 2, it will take at most 5 logic hops and 5

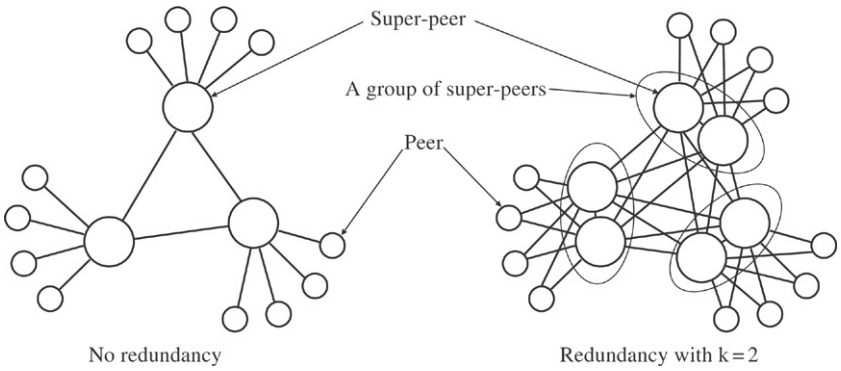

Fig. 4.3. Architecture comparison with redundancy and no redundancy.

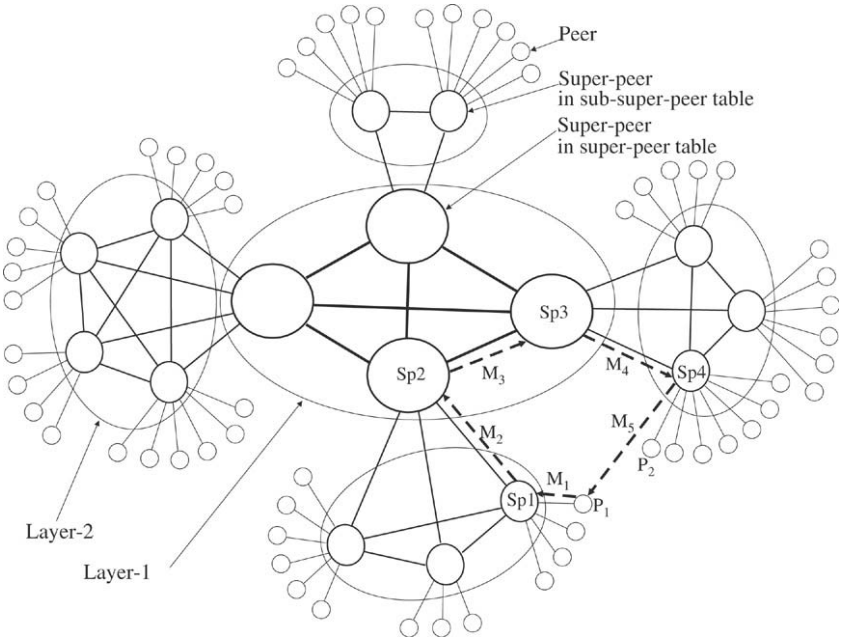

Fig. 4.4. Two super-peer layers.

messages to obtain the address of a peer who may have stored a needed file. In an ideal case, each super-peer in layer 1 will serve about 2000 super-peers, and each super-peer in layer 2 will serve about 1000 super-peers and 1000 peers; i.e., there will not be any super-peer that has to serve 10^6 peers or super-peers.

5. Related works

There are many works which have reported to how to improve the searching efficiency, decrease the number of transmitted messages and increase the adaptability of peer-to-peer systems to large-scale networks.

KaZaA [8] and Gnutella [23,25] have explored using heterogeneity of peers to improve search performance. These systems have efficient peers hold more neighbors and process more queries. An efficient peer (super-peer) acts as a server in a local area, builds an index of the shared files provided by those peers connected to it and offers a searching index service for those who have connected to it by flooding queries to other super-peers. These systems mainly explored improving performance by decreasing the number of transmitted messages and latency hops.

In [9] Yang et al. analyzed how to bring a balance between the efficiency of centralized searches, and autonomy, load balancing and robustness to attacks provided by distributed searching, through using super-peers.

In [24], Garces-Erice et al. studied hierarchical DHTs, in which peers are organized into groups, and each group has its autonomous intra-group overlay network and lookup service. The groups themselves are organized in a top-level overlay network. To find a peer that is responsible for a key, the top-level overlay first determines the group responsible for the key; the responsible group then uses its intra-group overlay to determine the specific peer that is responsible for the key. They concluded that hierarchical organization could improve a system's scalability. A hierarchical system demonstrates better stability due to selection of peers who are more reliable as

the members of the upper overlay, generates fewer messages in a wide area and can significantly improve the lookup performance by transmitting queries through the upper overlay.

Nejdl et al. proposed a design organizing super-peers with a hypercube structure in [26]. In their approach, every super-peer serves a subset of peers and all super-peers are arranged in a hypercube topology. Because the topology is vertex-symmetric, it features inherent load balancing among super-peers. When a super-peer wants to transmit a query, according to the spanning tree algorithm, it forwards the query to its neighbors instead of flooding the system with queries. Each super-peer wants to maintain at most d neighbors' information and it takes at most d logic hops for a query from any super-peer to the farthest super-peer, where d is the dimension of the hypercube.

Chord [6], CAN [16], Pastry [17], Tapestry [19] etc. use a flat overlay and exploit a DHT technique to structure all peers. To quickly locate data, data are mapped with keys associated with peers and each peer is responsible for a subset of the keys by a certain algorithm. During searching, the system does not need to be flooded with queries. The maximal latency and number of transmitted messages for a query vary with the number of all the peers in a logarithmic level.

In the Kelips system [11] Gupta et al. divided all peers into k virtual affinity groups, by using a consistent hash function mapping the peer's identifier into the interval $[0, k-1]$. Besides knowing the inner peers' information and filetuples, each peer in a group has to maintain a contact list which includes a small (constant-sized) set of peers' information of every other affinity group. Using those contact peers, the Kelips system can provide $O(1)$ time and $O(1)$ message complexity in the file lookup operation. To maintain this case, the number of routing entities that each node needs to save is in proportion to $n^{1/2}$. Here n is the number of peers in the system. Its aim is to reduce file lookup times and increase stability to failures and churn with increased memory usage and constant background communication overheads.

In [12], to achieve scalability and high performance, Freedman et al. introduced three-level rings of distributed sloppy hash tables (DSHTs) called clusters, based on the Chord ring. According to a uniform hash mapping rule, the external ring consists of all peers in the system. The middle ring consists of those peers whose round-trip time thresholds are 100 ms to each other. The inner ring consists of those peers whose round-trip time thresholds are 30 ms to each other. In all ring clusters, a peer maintains the same identifier. To insert a key/value pair, a peer performs a put-operation on each level of its clusters. To retrieve a key, a requesting peer first performs a get-operation in its inner ring cluster. If this operation fails, it uses the returned routing information to continue the query efficiently in an upper-ring cluster. Its main goal is to improve the efficiency of querying by improving the latency of lookup queries rather than reducing the hop count.

In [27], Mizrak et al. proposed a design based on the Chord ring. In their approach, there are two rings, named the inner-ring and outer-ring respectively. Each peer is placed on a circular identifier space in the "outer-ring", using a DHT algorithm such as Chord. Of all the peers, m peers who joined the system first are selected as super-peers to create a smaller core "inner-ring". The outer-ring is divided into m equal arcs and each arc is assigned to one super-peer. Each super-peer is responsible for maintaining two pieces of information: the addresses of the peers contained within its arc and the mapping between arcs and their responsible super-peers. Each peer registers in only one super-peer, and requests searching services from its super-peer. Each super-peer offers searching services for its registered peers and the other super-peers. The lookup is performed using super-peers in constant time. When a super-peer's load approaches its capability, it may share part of its load with its neighbors if they have sufficient excess capacity or with a new super-peer selected from the volunteer peers. In either case the super-peer splits its arc appropriately and reassigns pieces of this range to the neighbors accepting the load.

6. Conclusions

We presented a structured hierarchical P2P network model to achieve balance among the lookup latency, the number of transmitted messages and the expansibility of networks. In our model, when searching an index, it takes no more than 3 logic hops and 3 messages in a middle-scale network or 5 logic hops and 5 messages in a super-scale network. The model has good expansibility, and is suitable for large-scale networks. Furthermore, our system is robust. If the number of failed super-peers is less than the redundancy number in any logic area, our system still works well in theory.

Acknowledgements

This work was supported by the National Key Basic Research and Development Project (973) of China (No: 2003CB314805).

References

[1] Napster, http://www.napster.com/.
[2] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, S. Shenker, Making gnutella-like networks scalable, in: ACM SIGCOMM, 2003, pp. 407–418, Gnutella, http://gnutella.wego.com/.
[3] B. Yang, H. Garcia-Molina, Comparing hybrid peer-to-peer systems, in: Proc. of the 27th Intl. Conf. on Very Large Databases, September 2001.
[4] D.M. Lewin, Consistent hashing and random trees: Algorithms for caching in distributed networks, Master thesis, Department of EECS, MIT, 1998. Available at the MIT library, http://thesis.mit.edu/.
[5] D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, R. Panigrahy, Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web, in: Proceedings of the 29th ACM Symposium on Theory of Computing, STOC, 4–6 May 1997, pp. 654–663.
[6] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, H. Balakrishnan, Chord: A scalable peer-to-peer lookup service for internet applications, in: Proc. ACM SIGCOMM 2001, 2001, pp. 149–160.
[7] I. Clarke, O. Sandberg, B. Wiley, T. Hong, Freenet: A distributed anonymous information storage and retrieval system, in: Proceedings of ICSI Workshop on Design Issues in Anonymity and Unobservability, Berkeley, California, June 2000, Freenet.sourceforge.net.
[8] KaZaA, http://www.kazaa.com/.
[9] B. Yang, H. Garcia-Molina, Designing a super-peer network, Technical Report, Stanford University, 2002. Available at http://www-db.stanford.edu/~byang/pubs/superpeer.pdf.

[10] I.J. Taylor, From P2P to Web Services and Grids, in: Peer in a Client/Server Word, Springer-Verlag London Limited, 2005.
[11] I. Gupta, K. Birman, P. Linga, A. Demers, R. Van Renesse, Kelips: building an efficient and stable P2P DHT through increased memory and background overhead, in: 2nd International Peer-To-Peer Systems Workshop, IPTPS 2003, Berkeley, CA, USA, February 2003.
[12] M. Freedman, D. Mazieres, Sloppy hashing and self-organizing clusters, in: 2nd International Peer-To-Peer Systems Workshop, IPTPS 2003, Berkeley, CA, USA, February 2003.
[13] S. Zhou, Study on the distributed routing algorithm and its security for Peer-to-Peer computing, Ph.D. Thesis, Chengdu, University of Electronic Science and Technology of China, 2004.
[14] TTL(Time To Live), http://www.fact-index.com/t/ti/time_to_live.html.
[15] R. Schollmeier, A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications, in: Proceedings of the first International Conference on Peer-to-Peer Computing, P2P2001.
[16] S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Shenker, A scalable content-addressable network, in: Proc. ACM SIGCOMM, San Diego, CA, August 2001, 2001.
[17] A. Rowstron, P. Druschel, Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems, in: Proc. of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001), Heidelberg, Germany, November 2001.
[18] R.L. Rivest, The MD5 Message Digest Algorithm, RFC1320, April 1992.
[19] B. Zhao, J. Kubiatowicz, A. Joseph, Tapestry: An infrastructure for fault-tolerant wide-area location and routing, Technical Report UCB/CSD-01-1141, Computer Science Division, U. C. Berkeley, April 2001.
[20] National Institute of Standards and Technology, NIST FIPS PUB 180, Secure Hash Standard, U.S. Department of Commerce, May 93.
[21] M. Ripeanu, A. Iamnitchi, I. Foster, Mapping the gnutella network, IEEE Internet Computing 6 (1) (2002) 50–57.
[22] T. Saroiu, P. Krishna Gummadi, S.D. Gribble, A measurement study of peer-to-peer file sharing systems, in: Proceedings of Multimedia Computing and Networking 2002, MMCN '02, San Jose, CA, USA, January 2002.
[23] K. Truelove, Gnutella and the transient web, Whitepaper, 2002.
[24] L. Garces-Erice, E. Biersack, P. Felber, K. Ross, G. UrvoyKeller, Hierarchical peer-to-peer systems, in: Proceedings of EuroPar, Klagenfurt, Austria, 2003.
[25] Q. Lv, S. Ratnasamy, S. Shenker, Can heterogeneity make gnutella scalable? in: Proc. IPTPS, March 2002.
[26] W. Nejdl, M. Wolpers, W. Siberski, C. Schmitz, M. Schlosser, I. Brunkhorst, A. Löser, Super-peer-based routing and clustering strategies for RDF-based peer-to-peer networks, in: Proceedings of the 12th International World Wide Web Conference, Budapest, Hungary, 2003.
[27] A. Mizrak, Y. Cheng, V. Kumar, S. Savage, Structured super-peers: Leveraging heterogeneity to provide constant-time lookup, in: IEEE Workshop on Internet Applications, 2003.

Hongjun Liu is a graduate student in the Department of Computer Science at Tsinghua University, Beijing, China. His current research interests are in network security, distributed computing and peer-to-peer systems. He received his bachelor's degree in silk engineering from Suchou Silk Engineering Institute in 1990.

Ping Luo received his Master degree from the Department of Mathematics, Xiangtan University, China, in 1989, and his Ph.D. from Institute of Systems Science, Academia Sinica, Beijing, China, in 1996. He is an associate professor at the Department of Computer Science, Tsinghua University. His research interests are computer arithmetic, cryptography and security of network information, elliptic curves cryptosystems, secure E-commerce, multi-body contact problems and variation inequalities, domain decomposition methods and parallel algorithms.

Zhifeng Zeng, female, born in Hunan Province, December 1963, associate professor, main research interests: computer network security technology, next generation internet technology.