Effective Load Balancing in P2P Systems

Zhiyong Xu
Suffolk University
zxu@mcs.suffolk.edu

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

***Abstract*—**
In DHT based P2P systems, various issues such as peer heterogeneity, network topology, and diverse file popularity, may affect the DHT system efficiency. In this paper, we propose an effective load balancing algorithm for DHT-Based P2P systems. Our main contributions are: (1) we propose an fully distributed mechanism to maintain the history of file access information. This information is used to predict the future file access frequencies and support the load distribution and redistribution operations; (2) we design a novel load balancing algorithm, which takes the file access history and peer heterogeneity properties into account to determine the load distribution. Our algorithm can generate the best load distribution decision when a new peer comes, it can also be able to dynamically perform the load redistribution during system running time if overloaded peers appeared. In our algorithm, no virtual servers are used, thus we have less processing overhead on the expensive routing metadata maintenance; (3) finally, we design a topologically-aware data replication mechanism, the topological information of the peers are used for file replication decisions. A file is replicated only on a peer close to the group of peers which have high access frequencies.

I. Introduction

Peer-to-Peer (P2P) model attracted many attentions from both industry and academic communities due to its promising properties such as potential load balancing, self-autonomous organization and efficient computer resources utilization. Various P2P applications appeared [1], [2], [3], [4]. It is commonly believed that the P2P model have good load balancing performance. However, a P2P system faces the following questions: How to divide the system workload? How to distribute the workload on the peers? The system performance is highly depended on the answers to these questions. An imbalanced load will cause long file retrieve latencies and hurt the system overall performance. Clearly, a well designed load balancing algorithm is in great need.

Structured P2P systems including CAN [5], Pastry [6], Chord [7] and Tapestry [8] presented a logically elaborated workload distribution mechanism. In the ideal situation, peers and files are evenly allocated, peers have the equal capability and the file access frequencies are equal as well. Thus, system workload is evenly distributed on all the peers. System is well balanced and is able to achieve the optimal performance.

However, such a perfect world is not realistic. In real life, DHT based P2P systems still have severe load imbalance problems [9]. In a large-scale DHT system, with the collision free peerid generating algorithm which generating identifiers randomly, the size difference of the smallest zone and the biggest zone could be in the order of $O(nlogn)$. Obviously, the peer who owns a larger zone has to take more duty than the peer with a smaller one. In addition, computer resources, such as computational power, storage capacity and network bandwidth are quite different among peers. Saroiu et al conducted a measurement study on Napster and Gnutella [10], they found that the magnitude of these resources could vary between three and five order of magnitude across the peers. If a big zone is allocated to a weak peer (a peer with insufficient resources), even with a uniform workload distribution, serious load imbalance problem may occur.

There are many other factors have the great influence on the system performance as well. For example, the file popularity. Access frequencies of files may vary in the three orders of magnitude. Chu, et al [11] found that the most popular 10% files account for almost 60% of overall network traffic. The sizes of files are also quite different. The length of the smallest file may contain tens of bytes only, while the biggest file could be several GB in length. Thus, different amount of computer resources are needed to serve different files. The locations of files may affect the system performance as well. Fetching a file from a remote peer consumes much more bandwidth, and result in a long retrieving latency. Thus, the overlay network topology has its own effects. Such a skewed file distribution needs to be well considered in load balancing algorithm design. However, all the current DHT load balancing algorithms neglect at least one or more factors and can not guarantee the best performance.

In this paper, we propose a new load balancing algorithm to solve the above problems. In our solution, file access history information is collected and stored on each peer. This information together with peer heterogeneity are used for making load distribution and redistribution decisions. The size of the zone each peer owns is dynamic, it can be changed during system running period. Furthermore, the overlay network topology and file access history information are used for file replication decisions. In summary, our contributions are as follows:

1) We design an efficient, distributed mechanism to collect the file access history information, this information is used to predict the future file access trend;
2) We propose a simple and accurate load distribution mechanism. When a new peer comes, the workload on the peer which owns the zone it will join is well divided. The size of the zone new peer owns is determined by its capacity and the file access history;
3) We design a low overhead load redistribution mechanism. As the system keeps running, the size of the zone owned by a peer may be adjusted to reflect the change of the workload characteristics;
4) For data replication, we propose a topologically-aware mechanism. By placing the copies of files on the peer close to the group of peers which have high possibility to access the file, our algorithm is able to achieve better load balancing, reduce bandwidth consumption, and decrease the user perceived access latency;

5) In our algorithm, no virtual servers are used. We avoid the induced heavy routing metadata maintenance overhead.

The rest of the paper is organized as follows: In Section II, we discuss the mechanism of managing the historical file access information and usage for file access behavior prediction. In Section III, we describe the system design, and the detailed load balancing schemas. We describe the topologically-aware file replication mechanism in Section IV. The simulation experiments design and performance evaluation are presented in Section V, and Section VI discusses the related works. Finally, we conclude the paper in Section VII.

II. File Access History Information

One drawback existed in almost all the P2P load balancing algorithms [12], [13], [14] is the negligence of the workload characteristic, which is the popularity of files. These algorithms generate the load distribution and redistribution decisions based on the instant load information on peers only. If a peer is overloaded, some of its virtual servers will migrate to another light-loaded peer. However, which virtual servers to be moved are determined by the loads and capacities of both peers, no file popularity information is used. Thus, system has no clue about the future user access behaviors on these files, and in most cases, failed to produce the optimal decision for the virtual server movements. Even worse, severe virtual server thrashing (a virtual server migrate back and forth) may occur.

To generate better load balancing decisions, estimate the future file access frequencies is essential. An effective future file access frequency prediction mechanism is in great need. In our solution, we analyze the file access history information and use it for future file access estimation. It has been proved that large amounts of localities exist in web services, and this locality is the basis for web caching techniques [15], [16], [17], [18]. Chu et al. [11] found that the file access locality is prevailing in P2P systems as well. A hot file which has been accessed many times recently is also likely to be requested by some other peers in the near future. Thus, keeping the past file access history information can help us to predict the future file access tendency.

We use a simple and efficient mechanism to maintain the file access history information. Like the routing metadata, it is not possible to keep all file access history information on a single peer. A fully distributed management strategy is necessary. In our solution, we borrow the idea used for the routing metadata management, each peer only maintain a small portion of all history information, and it is only responsible for maintaining the access history information of files whose fileids are fallen into its zone. This information can be used for static load distribution when a new peer comes, it can also be used for load redistribution and data replication decisions.

Figure 1 shows a sample file access history information table (History Table) stored on the peer with the peerid 121. This peer owns the zone (117,121] on the name space. Thus, it has to maintain the access history for files whose fileids are 118, 119, 120 and 121. For each file, a LRU list is created to record the peers who accessed it recently. The peerid of the peer which generated the most recent request is stored on the header of the list. For each entry, the time that access occurred is also recorded. (Omitted in Figure 1). In this sample, File 118 has been accessed by the peers 223, 197 and 037 recently, among these accesses, peer 223 generated the most recent request.

As the P2P system keeps running, the amount of file access history information is increasing rapidly. Due to this large amount of records of file accesses, history table may need more storage space. Furthermore, the LRU operations on this table require more system resources and could affect normal tasks. On the other hand, maintaining file access history for a long time is not very useful. For a given file, the lifetime access curve is highly skewed. For example, the scoreboard of a NBA game is very hot if the game was finished a moment ago, and it can attract a significant amount of concurrent requests. However, it will become outdated after one week, and most users will not access it anymore. Thus, in history table, we only need to keep each file's access records for a certain period. All the access history before that period has no much usage, and should be removed.

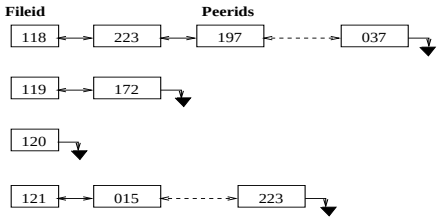

Fig. 1. A sample History Table on the peer 121

III. Load Balancing Mechanism

In this section, we describe the detailed load balancing algorithm. The principle of our strategy is to utilize file access history information to estimate the future file access frequencies. Our algorithm contains two components: a static load balance algorithm which is used when a new peer joins, and a load redistribution algorithm which can be used when a peer leaves the system or one peer becomes overloaded.

A. Determine the Best Workload Distribution

First, we create a network model to describe the load balancing problem in DHT-based P2P systems. Assume there are N peers in a DHT system, their capacities are $C_1, C_2, ..., C_n$, the zones owned by these peers are $(Z_{l1}, Z_{h1}], (Z_{l2}, Z_{h2}], ..., (Z_{lN}, Z_{hN}]$, and the workloads on these peers are $W_1, W_2, ..., W_N$, respectively. The system contains M files distributed on these peers, they have the different lengths. Thus, they have different access overhead. For these files, we assume the corresponding access overheads are $O_1, O_2, ..., O_M$, and the access frequencies are $F_1, F_2, ..., F_M$. Under the ideal situation, system workload should be well distributed on the peers proportional to their capacities. Thus, for a peer K, we should have:

$$\frac{W_K}{\sum_{i=1}^{N} W_i} = \frac{C_K}{\sum_{i=1}^{N} C_i} \tag{1}$$

We estimate the workload on a peer K as the sum of access frequencies for all the files whose fileids are fallen into $(Z_{lK}, Z_{hK}]$ in the past history information recording period. Assume those files are a1, a2, ... , ae. We have the following formula:

$$W_K = \sum_{x=a1}^{ae} F_x O_x \tag{2}$$

and

$$\sum_{i=1}^{N} W_i = \sum_{j=1}^{M} F_j O_j \quad (3)$$

Thus, we have,

$$\frac{\sum_{x=a1}^{ae} F_x O_x}{\sum_{j=1}^{M} F_j O_j} = \frac{C_K}{\sum_{i=1}^{N} C_i} \quad (4)$$

The above formula shows that to achieve the perfect load balancing, the ratio of the workload on a given peer to the overall system workload should be proportional to the ratio of its capacity to the aggregated capacity of all peers. In such case, the workload is well distributed on all the peers according to their capacities.

However, in reality, such a flawless workload distribution is impossible to produce due to the following reasons: (1) we can not obtain the accurate file access frequency information; (2) A DHT based P2P system is fully distributed, no peer can have a global view of the whole system, and no peer knows all other peers as well as their workloads. It is hard to calculate the overall workload distribution by using the above formula. To keep the system in the perfect balanced workload distribution will incur significant overhead; (3) A DHT based P2P system is dynamic, the file access behaviors change from time to time. A perfect workload distribution at a certain time may not be the best solution at another time.

On the other hand, the ultimate purpose of load balancing is to achieve the lowest average user perceived file access latency. It is not necessary to have the perfect load distribution. According to the queuing theory, in case a server is not overloaded, the average response time increases very slowly as we add more tasks on it. However, if the server is overloaded, the average response time will increase drastically only if we keep adding more tasks.

From the above discussion, we conclude that in DHT systems, it is not possible and not imperative to achieve the optimal load balancing in the global level. Our solution aims to provide the best load balancing property for small regions (each region contains a small number of peers). We believe if we can guarantee the optimal workload distribution for each small region, we can preserve relative good load balancing property for the whole system.

In our load balancing mechanism, we use the history table to estimate the future file access frequency. The algorithm contains two steps: First, when a new peer comes, we try to achieve the best static workload distribution assignment as we can; Second, during the system running time, we perform the workload adjustment operation only when it is necessary (say, a peer is overloaded).

B. Static Load Distribution

When a new peer P comes, the static load distribution algorithm is executed. The objective is to balance the workload on P and the peer T who owns the zone P will join. The division of the workload previously taken on T is determined by many factors, such as the capacities of peers T and P, the estimated future workload in the zone owned by T, etc.

In traditional DHT systems, when a new peer comes, a unique identifier (peerid) is assigned. In general, the peer's unique information such as IP address or the network adapter card address can be used as the seed for this generation. This peerid is used to determine the numerical position on the name space for this peer. Assume the new coming peer P is given the peerid p, to create the routing structure and other metadata information, P has to contact a nearby peer and ask it to use the standard DHT routing algorithm to search the peer who is in charge of the zone P belongs to. Let's suppose this peer is T, and the zone it owns is (n1, n2]. As shown in Figure 2, after the new peer is added, this zone will be divided into two smaller zones (n1, p] and (p, n2]. Peer P owns the first zone and T owns the second one. Clearly, no workload character is considered, system divides the zone based on the peerid p generated by SHA-1 algorithm.

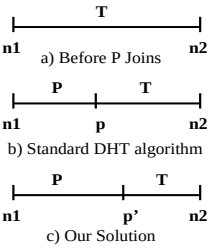

Fig. 2. Sample Peer Join Operation

Such a natural and simple zone allocation mechanism does not perform very well. It can not guarantee that the workload in the zone previously owned by P is well distributed on these two peers. After the division, it is very likely that one peer takes most of the previous workload and another peer does not have any workload to take at all. Severe load imbalance may occur. In our solution, we solve this problem. We argue that the purpose of generating the peerid with a collision free algorithm is to avoid the peerid conflicts and find the zone the peer to join. For the new peer P, after this zone is found, it is not compulsory to use p as the peerid. We can choose another position p' (within this zone) which can better divide the workload. To implement this, in our approach, both the historical file access information and the capabilities of these two peers are used to make the zone division decision. Figure 2 shows an example. The new peer P is given the peerid p', other than the value p. After the join operation finished, P owns the zone (n1, p'] and T owns the zone (p',n2].

The peer join algorithm is shown in Figure 3. Assume the capacities for peers P and T are C_P and C_T. Before the peer join operation is executed, for peer T, the fileids which are fallen in the zone (n1, n2] are a1,a2,...,ae, and the corresponding access frequencies of these files (obtained from file access history table) are $F_{a1}, F_{a2}, ..., F_{ae}$. Thus, the aggregated workload in (n1, n2] is $W_T = F_{a1} * O_{a1} + F_{a2} * O_{a2} + ... + F_{ae} * O_{ae} = \sum_{i=a1}^{ae} F_i O_i$. To well divide W_T, we need to find a number u, such that $a1 < u < ae$, and we have:

$$\frac{C_P}{C_T} = \frac{\sum_{i=a1}^{u} F_i O_i}{\sum_{j=u+1}^{a} eF_j O_j} \quad (5)$$

After this particular value "u" is found, we can randomly choose a position p', such that $u < p' < (u+1)$, as the peerid of the peer P, instead of p.

Since the previous workload W_T on T is divided into the new workloads W'_P on P, and W'_T on T. We have the following formulas,

$$W_T = W'_P + W'_T \quad (6)$$

Where,

P is the new coming peer
T is the peer who is in charge of the zone P will join
C_P is the capacity of P
C_T is the capacity of T
Function PEER_JOIN(P)
{
p = SHA-1(P);
T = DHT_Routing(p);
Determine the zone (n1, n2] for peer T
Find the access history information for files a1,a2,...,ae
$W_T = \sum_{i=a1}^{ae} F_i O_i$
$W'_P = \frac{C_P}{C_P+C_T} W_T$
i = a1
s = 0
while ($s < W'_P$)
{
$s = s + F_i O_i$
i = i + 1
}
p'= random(fileid(i), fileid(i+1))
Migrate files within (n1,p'] from T to P
Move file access history records in (n1,p'] to P
return(n1,p',n2);
}

Fig. 3. Static Load Balancing Algorithm

$$W'_P = \sum_{i=a1}^{u} F_i O_i \tag{7}$$

$$W'_T = \sum_{j=u+1}^{ae} F_j O_j \tag{8}$$

Thus,

$$\frac{W'_P}{W'_T} = \frac{C_P}{C_T} \tag{9}$$

Then, the previous workload W_P is well distributed on peers P and T. For the file accesses to be happened in the near future, we cam expect the workload should be able to be well allocated to these two peers.

Suppose there's no peer is overloaded before the peer join operation occurred, no peer will be overloaded after the peer join operation finishes, because in this operation, we do not add any additional workloads on any peers except the new peer. Even for the new peer P, no workload more than its capacity will be assigned.

Currently, the algorithm is performed on two peers, P and T, only. We can easily extend our algorithm by using more existing peers for the static workload distribution. It can achieve better local load balancing because more peers are involved. However, it will also bring more computational and file movement overheads. In our simulations, we evaluate their effects.

In case of a peer leaves the system, the zone previously allocated to it will be combined with the zone owned by its successor or predecessor. No further operation is needed. However, such a strategy may cause load imbalance on that peer in the future. To relieve this problem, we can divide the files previously taken on the leaving peer to its adjacent two peers. Each peer only takes a portion of the leaving peer's workload. If any of them get overloaded, the following load redistribution algorithm can be used.

C. Load Redistribution

In the above peer join algorithm, static zone division mechanism is used to guarantee the best load division at the time the new peer joins. However, the file access frequencies always change from time to time. It is very common that, after a certain amount of time, a hot file with high access frequency become very cold and no peers request it any more. In contrast, an originally cold file may become very hot. Furthermore, in case a peer leaves the system, its successor who has to take all its workload may easily become overloaded. If we don't dynamically redistribute the workload during the system running period, the system performance will be degraded gradually.

On the other hand, the workload redistribution involves future file access prediction and data movement operations. It may introduce significant amount of overhead. In our algorithm, we conduct load redistribution process only if it is necessary. That is, one or two peers get overloaded. This could happen when a peer leaves the system, and its successor is overloaded or the change of file access behavior cause a peer overloaded gradually.

For each peer, we define an overloaded valve V_o. For example, assign V_o to 80%. It means, if the workload allocated on a peer is over 80% of its capacity, the response time for the requests will increase significantly, and this peer is considered as overloaded. The load redistribution operation has to be executed. We also define another safe valve V_s. For example, V_s is 50%. When we conduct the load redistribution, the goal is to make workloads on all the participating peers under V_s of their capacities to guarantee none of them will again get overloaded very soon.

Figure 4 shows the load redistribution algorithm. Assume the peer K is overloaded. It will start a load redistribution operation. It contacts its immediate successor L_1 and predecessor J_1. Assume, their corresponding workloads are W_K, W_{L_1}, and W_{J_1}, respectively. The aggregated capacity of these three peers $AC = C_K + C_{L_1} + C_{J_1}$. If the sum of workloads on K, L_1, and J_1 is larger than $V_s * AC$, then more peers have to be involved. Peer L_1 will contact its successor L_2, and peer J_1 will contact its predecessor J_2. If the aggregated workload $AC = C_K + C_{L_1} + C_{J_1} + C_{L_2} + C_{J_2}$ is still above $V_s * AC$, more peers will be contacted. If the sum of the workloads is less than $V_s * AC$, the load redistribution operation will be conducted on the set of current peers. The workload within these zones will be rearranged according to peers' capabilities and the future file access tendencies. A variance of the algorithm is, in each round, instead of adding a successor and a predecessor, we may include only one of them. Then, we can reduce the amount or introduced overhead. However, the basic ideas for these two schemes are the same.

Figure 5 shows an example. Before the peer K leaves, the zones owned by the peers P0, K, P1 and P2 are (n1,t1],(t1,t2],(t2,t3] and (t3,n2], respectively. After K left, P1 is overloaded, the load redistribution operation has to be enforced. After the operation finishes, the zones owned by peers P0, P1 and P2 become (n1,t1'],(t1',t3'] and (t3',n2], respectively. Such a distribution is based on the capacities of P0, P2 and P3 as well as the file access history information within (n1,n2].

IV. Topological-aware File Replication

Besides the above mechanisms, we can further improve the load balancing performance in DHT system by making copies

K is the peer which is overloaded, it owns the zone (n1, n2]
C_K is the capacity of Peer K
W_K is the current workload on K
V_o is the overloaded valve
V_s is the safety valve
Function LOAD_REDISTRIBUTION(K)
{
if ($W_K < V_o * C_K$)
return
L_1 = successor(K)
J_1 = predecessor(K)
$W = W_K + W_{L_1} + W_{J_1}$
$L = L_1$
n2 = the upper bound of the zone Peer L_1 owns
$J = J_1$
n1 = the lower bound of the zone Peer J_1 owns
$AC = C_K + C_{L_1} + C_{J_1}$
while ($W > V_s * AC$)
{
L = successor(L)
n2 = the upper bound of the zone L owns
J = predecessor(J)
n1 = the lower bound of the zone J owns
$W = W + W_L + W_J$
$AC = AC + C_L + C_J$
}
Find all the files a1,a2,...,ae within (n1,n2]
x = 1
N = J
while ($N <= L$)
{
s = 0
$W'_N = \frac{C_N}{\sum_{i=J}^{L} C_i} W$
while ($s < W'_N$)
{
$s = s + F_x O_x$
x = x + 1
}
t' = random(fileid(x), fileid(x+1))
Migrate files to N
Move file access history to N
N = successor(N)
}
}

Fig. 4. Workload Redistribution Algorithm

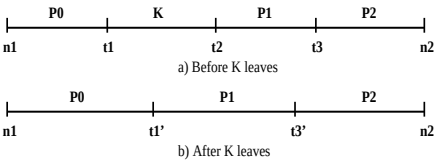

Fig. 5. A Sample of Load Redistribution

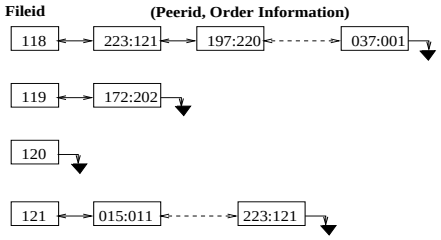

Fig. 6. A Modified History Table on Peer 121:012

of popular files on multiple peers. Previous DHT systems also use this strategy to enhance the data availability. However, in our approach, we take the peers' network topological characteristics into account. We make the copies of a file on the peers topologically adjacent to the group of peers which are likely to access this file in the future. Thus, we can achieve better load balancing, reduce the user perceived latency and the network bandwidth consumption. We use file access history information to make the data replication decisions. In this section, we first introduce the technique to discover peers' topological information, and then we describe the modified history table, finally we present the new data replication mechanism.

A. Distributed Binning Scheme

To create topologically-aware data replication schema, the first question we should answer is how to represent and maintain the network topology information. For example, how to discover the topologically close peers for a given peer? Clearly, we need some mechanisms which can represent the topological location information of the peers. A simple method for this purpose is the distributed binning scheme proposed by Ratnasamy and Shenker [19].The detailed information about the distributed binning scheme can be seen in [19].

We use the landmark node ordering information as part of the peer identification information. For example, Peer 121:012 represents the peer with the peerid 121. Three landmark nodes (L1, L2 and L3) are used, and the link latencies from peer 121 to L1, L2 and L3 are within [0,20), [20,100) and greater than 100ms, respectively. The peers who have the same or similar ordering information are topologically close. For example, Peer 121:012 is topological closer to Peer 206:012 than Peer 124:202, which means the link latency to the Peer 206:012 is much smaller than to the Peer 124:202.

B. Modified File Access History Table and Topology Table

We modify the structure of the file access history table to include the order information. Figure 6 shows the sample table on Peer 121:012. In this table, every entry contains three components: the peerid, the order information of that peer, and the access time. Again, the last item is omitted.

We create another hash table (Topology Table) to represent the aggregated number of accesses coming from peers with the same order information. A sample table is shown in Figure 7. For each entry in the table, it contains two components, the first item is the order information, and the second one is the number of file accesses in the past recording period. For an entry with the value (t,n), it represents, that in the past counting period, there's *n* accesses to the file came from the peers which have the common order information *t*. For example, look at the first entry

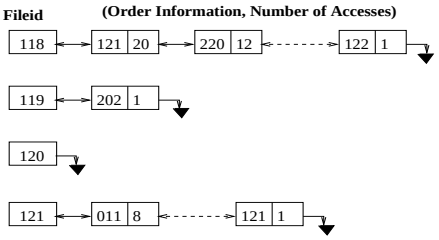

Fig. 7. A sample Topology Table on peer 121:012

for the file 118, in the past period, there are 20 accesses came from the peers with the order information 121, but only 1 access came from the peers with the order information 122.

C. File Replication Mechanism

We propose a new file replication mechanism to further improve the load balancing performance by making multiple copies of hot files on different locations. Here, the modified history table is used. For each entry in this table, the order information indicates the topological information of the peer who requested this file. We can detect if a file is very hot or not by checking the records in the topology table, and find out the most common order information which have the largest number of requests. Then, we search the records in the history table and choose some peers who have that common order information. These are the peers topologically adjacent to the group of peers that have the highest requests. We predict more request will come from this group in the future. A copy of the file can be created on one of those peers. Next time, if a new request for this file comes from a peer in this group (it can be detected from its order information), the client who initialize this request will be notified the location of the replica, and it can retrieve the file from a neighbor peer directly. With this strategy, we can greatly reduce the user access latency and decrease the Internet bandwidth consumption. For example, as shown in Figure 7, there are 20 accesses came from the peers with the order information "121". We can choose one of these peers and create a replica of file "118" on it. In the future, if another peer with the same order information requests this file, it can be redirected to the neighbor peer who has the copy.

To choose an adequate peer to store a copy, first, the peer P who is in charge of this popular file has to contact those peers by sending a message. The peers being selected reply with their estimated workload (using the file access history information), the peer R who has the lowest workload will be chosen to store a copy of that file. A direct link to that replica is created on peer P. Next time, if P observes a new request from a peer T which is topologically close to peer R, T will be informed and the request will be forwarded to R.

Such a strategy is still coarse, it does not exploit the full topological information. For example, peer "105:012" is in charge of a file F. F has a copy on peer "123:221". If a new request comes from peer "047:220", should we forward the request to "123:221"? A mechanism which can find the topological distance between peers with different order information is needed. One possible solution is to create some "super order information". They can be used to combine several order information together when making the file replication decision. For example, in Figure 7, peers with the order information 120, 121 and 122 can be combined to form a super order information "12X". Assume a copy of a file is stored on a peer with the order information 120, if a new request comes from a peer with order information "121", we should forward this request, because it has the same super order information with the peer who has a copy.

V. Performance Study

In this section, we evaluate the performance of our load balancing algorithm and compare it with previous solutions. First, We describe the simulation environment we use in our experiments, then we present the simulation results.

A. Experimental Environment

In our simulations, we choose GT-ITM Transit-Stub (TS model) as the primary network topology model. TS is an internetwork topology model proposed by E. Zegura in [20]. In our simulation, the delays of intra-transit domain links, stub-transit links and intra-stub domain links are set to 100, 20 and 5ms respectively (We also use other distributions but the tendency does not change). In all the experiments, the number of peers in simulated network is 5000.

We use the web proxy logs obtained from the National Laboratory for Applied Network Research (NLANR) as the workload. The trace data we use are collected from eight individual servers between April 30, 2003 and May 6, 2003. We vary the number of files in the system from 20K to 100K.

Four different load balancing strategies are compared: 1) **NLB**: this is the basic DHT system, with no load balancing strategy enforced. 2) **VSNM**: Using virtual servers, but server migration is not allowed; 3) **VSM**: Using virtual servers, and the server migration is used to dynamically balance the load; 4) **HLB**: This is our new algorithm: historical information based load balancing. The performance metric we use is the standardized Load/Capacity ratio. For each experiment, we calculate the average Load/Capacity ratio and normalize it as 1.0.

B. Load Balancing

Our first set of experiments evaluates the load balancing performance. Figure 8 shows the 75-percentile distribution of the aggregated workload/Capacity ratio for all the algorithms. The result can be used to represent the workload variances on the peers. The smaller the difference, the better the load balancing performance. From the figure, we can draw the conclusion that HLB algorithm outperforms all the other three mechanisms. The variance in HLB is the smallest, and it is about 65% less than NLB and VSNM, and around 25% superior over VSM. It is not exceptional that NLB has the worst performance. But surprisingly, VSNM can not achieve good performance either. This is because VSNM uses a static workload allocation mechanism, it does not consider workload characters, and virtual servers can not migrate if a peer is overloaded. Compare to NLB, not too much benefits we can get by using virtual servers. For VSM, since it dynamically redistributes the workloads, it can achieve better performance than NLB and VSNM. But it is still worse than HLB because of the lack of the consideration of the historical information.

From Figure 8, we can also observe that, as we increase the number of files in the system, for all the algorithms, the variance becomes larger. This is because file accesses are highly skewed, introducing more files will result in more imbalanced distribution of files, and exacerbate the existing problems. However, HLB is the best. The reason is, with the more files, we have

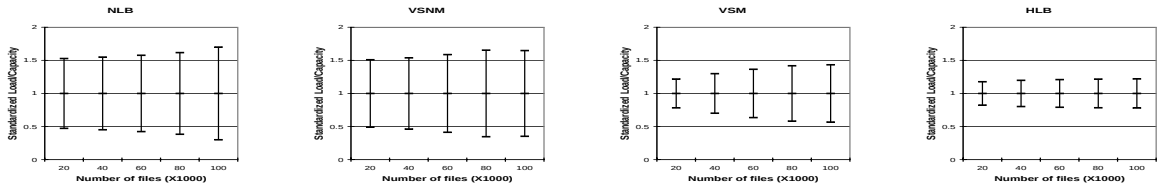

Fig. 8. 75-Percentile Distribution of Load/Capacity Ratio for Different Algorithms

more file access history information can be used to guide the load distribution. As the number of files increases from 20K to 100K, the variance in HLB only increases by 10%. For other algorithms, the variance increase by 19% to 35%. Clearly, predicting the future file access trend is helpful to achieve the optimal load balancing.

C. Overloaded Peers

In all four algorithms, a peer may got overloaded due to various factors, such as the file access behavior, peer heterogeneity, etc. In this set of experiments, we check the overload probability. A peer is considered as overloaded, if at a certain time, the workload assigned to it is above 80% of its capacity. The experimental result is shown in Table I. As the number of files in the system increases, more requests are used, and the system became busier. For all four algorithms, the percentage of overloaded peers increases as more files and requests are entered. For NLB algorithm, even with the smallest set of files (20K), it still has very bad performance, 7.23% of all peers got overloaded at least once. VSNM also have very bad performance, due to inflexibility to adapt the changed workload distribution. VSM performs much better than these two algorithms, because it will dynamically move the virtual servers to reflect the distribution. But the migration of virtual servers is blind, no file popularity information is considered. Thus it does not achieve the best performance. HLB has the smallest number of overloaded peers in all experiments, only 6.91% of all peers is overloaded with 100K files.

TABLE I

PROBABILITY OF OVERLOADED PEERS COMPARISON

	20K	40K	60K	80K	100K
NLB	7.23	10.15	15.37	18.23	21.82
VSNM	7.11	10.78	14.12	16.17	19.22
VSM	4.12	8.27	11.37	13.09	15.24
HLB	0.18	1.27	2.34	4.27	6.91

D. File Replication Performance

Finally, we test the effects of the topological-aware file replication mechanism. In this experiment, we only make two replicas for a popular file. We calculate the average access latency for the 100 most popular files. The result is shown in Figure 9. Since HLB has more information about the peers, a copy can be created on a peer topologically close to the peers who likely to access the file. For NLB, VSNM and VSM, no copies are made. HLB has the best performance in all the cases. We normalize its access latency as 1. All the other three algorithms have the worse performance than HLB, but none of them superior than other two. In our experiment, even if we create another copy for each file in NLB, VSNM and VSM, they have no clue which peer should be used to keep a copy. We did not see any performance advantages. Clearly, lack of the topologically information restrains the system ability to improve the load balancing performance.

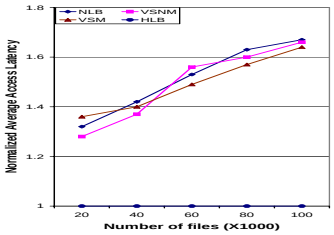

Fig. 9. File Replication Effects

VI. RELATED WORKS

A bunch of research papers [12], [13], [14] proposed dynamic load balancing algorithms to relieve the load imbalance in Chord. In all these approaches, virtual servers are used, and the peer heterogeneity is well considered to achieve a better address-space distribution. System dynamically monitors the workload distribution, and moves virtual servers from overloaded peers to light-loaded peers. However, the usage of virtual servers greatly increases the amount of routing metadata needed on each peer and causes more maintenance overhead. Furthermore, peers' topological characters are not taken into account.

On the other hand, even with the perfect address-space assignment, which means the sum of segments on all peers are proportional to their capabilities, the above systems may still not be able to achieve the optimal load balancing. This is due to the highly skewed file access behaviors. The fileid of a hot file may fall into a zone owned by a weak peer, and it is not capable to deal with the high demands for this file. It is also very likely that many files which are seldom used fall in a zone owned by a powerful peer. The computer resources on the powerful peer are wasted. The system workload is imbalanced, and the performance is limited.

In [21], Byers et al. proposed a simple "the power of two" load balancing strategy. In their approach, multiple hash functions are used to generate the fileids. In case a new file is inserted, it will be given multiple identifiers, the file will be stored on the peer who is the least loaded. Such a strategy is simple and efficient. However, it has two drawbacks: first, it increases the computational overhead for routing requests, since multiple hash functions have to be computed each time; Second, it is a

static allocation, and does not change in case the workload distribution shifts.

In [22], [23], the topological information was used to reduce the routing latency as well as the user retrieving overhead. However, load balancing issue is not considered. In this paper, we adopt this idea, and use topological information to improve DHT system load balancing performance.

None of the above systems considered the file access frequencies, and failed to predict the tendency of the future user access behavior. To the best of our knowledge, our paper is the first to utilize this information to improve load balancing performance in DHT system.

VII. Conclusions

In this paper, we present a novel and efficient load balancing schema for DHT-based P2P systems. We analyze the issue carefully, and figure out that the load balancing performance in DHT systems is highly affected by system properties such as the peer heterogeneity, file access behaviors, and P2P overlay network topology. In our solution, all these issues are well considered to generate better load balance decisions. Firstly, we create an efficient mechanism to maintain the file access history information, and use this information to predict the future file access behaviors. Secondly, by utilizing this information, we can more accurately split the workload when a new peer joins the system; Thirdly, by utilizing this information, we also create better load redistribution decisions by modifying the sizes of zones. In our algorithm, the load redistribution is only performed when it is necessary, the induced overhead problem is minimized. Fourthly, topological characteristics of peers are used to assist the file replication operations. Files are copied on the peers adjacent to the groups of peers which have high probabilities to access these files in the future. With this strategy, load can be well balanced on multiple peers, the user perceived retrieving overhead is reduced, and the bandwidth consumption is decreased as well. Finally, no virtual servers are used in our algorithm, we avoid the additional routing metadata maintenance overhead.

VIII. Acknowledgement

This research has been supported by NSF Grant CNS-0509207 and a research grant from Intel Corporation.

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] 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.
[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] 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.
[8] 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.
[9] V. King and J. Saia, "Choosing a random peer," in *Principles of Distributed Computing(PODC)*, (Newfoundland, Canada), July 2004.
[10] S. Saroiu, P. K. Gummadi, and S. D. Gribble, "A measurement study of peer-to-peer file sharing systems," in *Proceedings of Multimedia Computing and Networking (MMCN), San Jones, CA*, Jan. 2002.
[11] J. Chu, K. Labonte, and B. N. Levine, "Availability and Locality Measurements of Peer-to-Peer File Systems," in *Proceedings of ITCom: Scalability and Traffic Control in IP Networks II Conferences*, July 2002.
[12] B. Godfrey, K. Lakshminarayanan, S. Surana, R. M. Karp, and I. Stoica, "Load balancing in dynamic structured p2p systems.," in *Proceedings of IEEE INFOCOM*, 2004.
[13] J. Ledlie and M. Seltzer, "Distributed, secure load balancing with skew, heterogeneity, and churn," in *Proceedings of IEEE INFOCOM*, 2004.
[14] A. Rao, K. Lakshminarayanan, S. Surana, R. M. Karp, and I. Stoica, "Load balancing in structured p2p systems.," in *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS), Berkeley, CA*, pp. 68–79, 2003.
[15] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, "Summary cache: a scalable wide-area web cache sharing protocol," *IEEE/ACM Transactions on Networking*, vol. 8, no. 3, pp. 281–293, 2000.
[16] A. Rousskov and D. Wessels, "Cache Digest," in *the 3th International WWW Caching Workshop*, (Manchester, England), June 1998.
[17] D. G. Thaler and C. V. Ravishankar, "Using Name-Based Mappings to Increase Hit Rates," *IEEE/ACM Transactions on Networking*, vol. 6, no. 1, pp. 1–14, 1998.
[18] B. Smith and V. Valloppillil, "Personal communications," February-June 1997.
[19] 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.
[20] 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.
[21] J. W. Byers, J. Considine, and M. Mitzenmacher, "Simple load balancing for distributed hash tables.," in *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS), Berkeley, CA*, pp. 80–87, 2003.
[22] 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.
[23] Z. Xu, X. He, and L. Bhuyan, "Efficient File Sharing Strategy in DHT-based P2P Systems," in *Proceedings of the 24th IEEE International Performance, Computing, and Communications Conference(IPCCC'05)*, (Phoenix, AZ), April 2005.