J Inf Process Syst, Vol.10, No.4, pp.628~645, December 2014
http://dx.doi.org/10.3745/JIPS.03.0020

ISSN 1976-913X (Print)
ISSN 2092-805X (Electronic)

Cost-Effective Replication Schemes for Query Load Balancing in DHT-Based Peer-to-Peer File Searches

Qi Cao* and Satoshi Fujita*

Abstract—In past few years, distributed hash table (DHT)-based P2P systems have been proven to be a promising way to manage decentralized index information and provide efficient lookup services. However, the skewness of users' preferences regarding keywords contained in a multi-keyword query causes a query load imbalance that combines both routing and response load. This imbalance means long file retrieval latency that negatively influences the overall system performance. Although index replication has a great potential for alleviating this problem, existing schemes did not explicitly address it or incurred high cost. To overcome this issue, we propose, in this paper, an integrated solution that consists of three replication schemes to alleviate query load imbalance while minimizing the cost. The first scheme is an active index replication that is used in order to decrease routing load in the system and to distribute response load of an index among peers that store replicas of the index. The second scheme is a proactive pointer replication that places location information of each index to a predetermined number of peers for reducing maintenance cost between the index and its replicas. The third scheme is a passive index replication that guarantees the maximum query load of peers. The result of simulations indicates that the proposed schemes can help alleviate the query load imbalance of peers. Moreover, it was found by comparison that our schemes are more cost-effective on placing replicas than PCache and EAD.

Keywords—DHT, Load Balancing, Load Reduction, Multi-Keyword Search, Replication

1. INTRODUCTION

Peer-to-Peer (P2P) file sharing systems have been widely used in recent years. In P2P systems (we use 'P2P systems' to denote 'P2P file sharing systems' in the rest of this paper), an index is a common and useful data structure for object (i.e., file) retrieval. Mainly, an index is a set of entries associated to a given keyword, and an entry is a triple table that contains keyword, object metadata and object location. Retrieval of indices can be managed through a centralized or distributed means. In hybrid P2P systems, such as Napster and eDonkey, index retrieval is realized by sending queries to a dedicated server (or a cluster of dedicated servers) that maintains a set of indices in a centralized manner. On the other hand, in pure P2P, index retrieval is realized by participating peers in a distributed manner. According to the control policy of indexing, pure P2P can be again classified into unstructured P2P and structured P2P. In unstructured P2P, such as Gnutella and FreeNet, indices will be retrieved by time-to-live (TTL) based

Manuscript received October 22, 2013; first revision January 14, 2014; accepted March 3, 2014.
Corresponding Author: Qi Cao (caoqi@se.hiroshima-u.ac.jp)
* Department of Information Engineering, Hiroshima University, 1-4-1 Kagamiyama, Higashi-hiroshima, 739-8527, Hiroshima, Japan. ({caoqi, fujita}@se.hiroshima-u.ac.jp)

controlled flooding over peers in the systems, which severely limits the scalability of the overall system.

Alternatively, most of structured P2Ps adopt a distributed hash table (DHT) to overcome the low scalability of flooding-based indexing schemes. In such systems, each object is typically attached to several keys, and each key is mapped to a unique point in a hash space. The hash space is partitioned among participating peers, and each peer is responsible for storing all of the indices of objects whose keys are mapped to a portion of the hash space associated to the peer. Each peer can efficiently locate a peer that is responsible for a given key by identifying a point corresponding to the key in the hash space and by routing a query message towards the identified point. DHTs are often referenced by their geometry of hash space, such as a ring [1], torus [2], and some hybrid [3,4].

To retrieve indices, multi-keyword search is a quite common query type, and it generally utilizes an intersection or union operation of indices that are obtained by single keyword (or key) search. More concretely, in a DHT-based P2P system, by giving a multi-keyword query initiated by a user, it will be parsed into multiple sub-queries, each of which corresponds to a keyword contained in the query. Those sub-queries are sent to other peers by using the DHT. The search result received by the user is an intersection or union of the indices obtained by each sub-query. In this process, since the preference of users concerned with keywords contained in queries generally follows a power-law distribution, such as Zipf's law [5], an access concentration of queries would probably happen at two kinds of peers. For example, 1) home peers that hold indices associated with popular keywords and 2) intermediate peers who are in the intersection of multiple routing paths towards 'hot' home peers. In the system, those two kinds of peers easily become a bottleneck in query processing. If there were several of these types of overloaded peers in the routing path for a given query, it would incur long index retrieval latency and consequently, would negatively influence the overall performance of object retrieval.

Index replication has a great potential for alleviating this issue. However, existing schemes have not explicitly addressed it or they incur high costs [6-8]. The management cost of index replicas (i.e., cost of copying and keeping index replicas) is very small. For example, suppose that an entry is approximately 500 B, an index that consists of 200 entries would be approximately 100 kB. On the other hand, the maintenance cost of index replicas may be high if we want to guarantee the consistency between an index and its replicas, since the index entries associated with a keyword would be continuously inserted or deleted. The degree of overhead for guaranteeing index consistency may be varied according to the need of real systems, and it is an open problem in this paper. Generally, the cost of index replication is evaluated through the number of replicas rather than the size of replicas (i.e., the maintenance cost is assumed to be far larger than the management cost). Instead of index replication, using a pointer would be an effective approach to avoiding the consistency issue of index replication, where a pointer is a pair of key (i.e., hashed keywords) and index locations. In other words, the pointer is a 'shortcut' between a search key and a home peer of the keyword.

Replica placement has a significant impact on minimizing the replication cost. The replica placement problem that focuses on optimizing one kind of resource in P2P systems is found to be NP-complete [9] (i.e., the optimal placement of replicas will always lead to exponential time algorithms). Alternatively, a lot of heuristic algorithms have been proposed to give approximate solutions for the problem. The majority of existing replication methods place replicas in the query paths to intermediate peers between requesters and home peers, which is referred to as path-based replication or probabilistic replication. PCache [7] and EAD [8] are two prominent probabilistic replica placement schemes. On the other hand, deterministic replication [10] has also been proposed, based on DHTs. In deterministic replication, each point in a hash space is associated with a set of other identified points. For example, if point *i* is associated with identified point *r*, then any (key, value) pair with point *i* should be stored at the peers responsible for points *i* and *r*. To realize the benefits of these approaches for minimizing the replication cost, it is necessary to instantiate particular schemes for different classes of DHTs, utilizing their routing characteristics.

In this paper, we focus on a class of tree-based routing DHTs (See Section 3.1 for the details). We will propose an integrated solution consisting of three replication schemes to alleviate the load imbalance of peers, while minimizing the cost, for these types of DHT-based P2P systems. The first scheme tries to bind the maximum response load received by each index to have it not exceeding a threshold μ. Under such constraint, an index with a popular keyword (i.e., popular index) will make more replicas. It does so to allow the routing load for the popular index to be highly reduced and so that the response load received by its home peer and replica peers can share the popular index. The second scheme places a predetermined number of pointers for each index with the aim of decreasing the number of index replicas generated by the first scheme. The third scheme controls the maximum query load of peers by operations of pushing or pulling index replicas. It does so in order to eliminate a bottleneck situation in the query processing in the system. The result of simulations indicates that all the proposed schemes can work in coordination in alleviating the query load imbalance of peers. Moreover, it was proved by comparison that our schemes are more cost-effective in placing replicas of index to other peers in the system than PCache and EAD.

The remainder of this paper is organized as follows: Section 2 outlines work related to this topic. Section 3 describes the details of the system model. Section 4 proposes three replication based query load-balancing schemes. The result of the evaluation is given in Section 5. Finally, Section 6 concludes the paper and presents future work that should be carried out.

2. Related Work

In general, load balancing is a key technology for avoiding a peer failing in performing its tasks in different kinds of distributed systems. Load balancing in P2Ps has been researched more than ten years since the appearance of P2P systems. Load balancing can be achieved by shedding load from heavily loaded peers (i.e., source peer) to lightly loaded peers (i.e., destination peer) via task migration or task replication. Unlike replication, migration implies that once task has been moved to a destination peer, it will be deleted at the source peer. In this section, we give a brief description of the history of load balancing schemes for P2P systems, especially for DHT based P2P systems.

2.1 Task Migration Based Approaches

At an early research stage, load balancing in DHTs is aimed at balance storage load of peers (i.e., the number of objects per peer). Many of the load balancing schemes are based on the notion of virtual servers [11-13]. Each peer is assigned several virtual servers, each of which is responsible for a portion of the ID space (i.e., hash space) of the DHT, and a load distribution is conducted by migrating virtual servers among peers.

Another part of load balancing schemes is conducted by controlling the location of objects or peers. Rieche et al. [14] proposed load balancing schemes based on moving objects between peers. In the proposed schemes, the ID space is divided into intervals. Peers within an interval are collaborative for all of the objects stored in the interval. If the storage load in an interval exceeds a threshold value, the following operations are conducted: for example, 1) the interval with an excessive load is split into two intervals if the interval contains more than $2f$ peers; 2) it move peers from other intervals to the overloaded interval to reach $2f$ peers and splits it as in the previous case; and 3) if there are no more than f peers within the interval the interval borders between the overloaded interval and its neighbor intervals can be shifted to balance the load. Karger and Ruhl [15] proposed a mechanism that allows lightly loaded peers to relocate to the ranges of the ID space associated with many objects, in order to share the load of the peers that are responsible for these particular ranges of the DHT space. More concretely, a lightly loaded peer requests the load of a randomly chosen remote peer and makes a load comparison between itself and the remote peer. If the load of the remote peer is higher, the peer will relocate to share the range of the ID space of the remote peer.

As an alternative approach, Byers et al. [16] proposed a simple, but efficient, load balancing mechanism (i.e., the power of two choices). When inserting an object into the system, the objects are generated into multiple hash values using multiple hash functions, each of which corresponds to a candidate peer for receiving the object. After retrieving the load of the corresponding peer candidates, the object will be stored at the peer with the lightest load. The power of two choices states quite a surprising result. For example, an exponential improvement in reducing the maximum load of peers by using two hash values was discovered.

At a later research stage, some of researchers focused on balancing routing loads by dealing with the skewed popularity of objects. Bianchi et al. [17] proposed an adaptive load balancing mechanism in Pastry (i.e., where replacing peers with a high traffic rate with peers with low a traffic rate in the routing tables if a peer becomes overloaded). Shen and Xu [18] proposed a load balancing scheme based on elastic routing tables. By resizing the out-degree (i.e., the size of routing table) of peers, thereby controlling the in-degree assignment of peers, it can balance the routing load of peers.

2.2 Task Replication Based Approaches

Most of replication schemes are concerned with object replication based on single keyword (e.g. object name) search. Object replication is widely used to improve object search efficiency [6-8]. Meanwhile, it also implicitly improves the load balancing of peers. This is because replicas in the intermediate peers can intercept the queries towards home peers of popular objects, which is due to the fact that the load of home peers is shared by replica peers, and a part of peers' routing load is also decreased.

Typically, replication can be categorized into three types; i.e., client-side replication, server-side replication and path replication. By given an object, client-side replicates the object close to requester; server-side replicates the object close to object owner and path replicates the object to the peers in the query path from a requester to the object owner. However, most of these methods either have little effect in reducing the average search hops or they have a cost of high overhead.

Ramasubramanian and Sirer [6] proposed an approximate replication solution for a Pastry based system. Replicas of a given object are placed on a set of peers who has the same digit-based prefix of peer ID. By estimating the popularity of each objects, different ranges of ID space for each objects is determined. Their solution is mainly proposed for a distributed Domain Name System (DNS), where each object is a static URL. The popularity of URLs is generally relatively stable, while popularity of objects in file sharing systems varies in time.

Rao et al. [7] proposed an optimal replication algorithm (referred to as PCache in this paper) for minimizing average search hops. The algorithm is based on the observation that peers with smaller overlay distance to object owner have a higher frequency of visits, thus, the replicas are suggested to place consecutively at predecessor peers. Each replica peer will later respond to searches for the object. However, their observation of query distribution over peers might not be true under the most common cases. For example, some of the predecessor peers that are close to the object holder might forward very few queries if the access pattern over requesters is skewed. Hence, the replicas in the PCache might not be fully utilized. They also claimed that in order to optimize the average search hops the number of replicas generated for an object should be proportional to the popularity of the object if the total number of replicas is predetermined. However, the popularity estimation for all objects is a very difficult task in a distributed environment. In order to improve the hit rate of replicas, Shen [8] proposed an efficient and adaptive decentralized (EAD) replication algorithm. In the algorithm, each peer records and updates the query traffic rate of each object, denoted by q_f. By determining a constant value T_q based on the average query rate in the system, if $q_f > T_q$, the peer will initiate an object replication request to the object owner. Such requests will be temporarily saved in a cache of the object owner during a certain period of time. If the object owner becomes overloaded, it will replicate an object to the peers in the cache according to q_f, in a descending order. EAD can highly improve the utilization rate of replicas. However, in the manner of load shedding, EAD may place replicas close to the object owner. Therefore, EAD has not fully utilized the replica for reducing search hops.

3. System Model

In this section, we describe a basic keyword search model based on a class of tree-based routing DHTs. We will also describe a skewed query model used in this paper.

3.1 Keyword Search Model

We used a class of tree-based routing DHTs defined in [19]. In such DHTs, paths from any requester peer to all possible home peers that stored indices are aggregated and the resulting topology is a tree. Many popular DHTs, such as Chord [1], Pastry [3] and Tapestry [4], belong to this category. More concretely, in this paper, we consider a Chord (an example of a tree-based routing DHT) based P2P file search system consisting of N peers, $n_i \in N$, $i \in \{1, 2, \dots, N\}$. In the system, the index entry of each object held by a peer is maintained by its corresponding peers. This mapping is controlled by a set of keywords attached to the object. By using the consistent hashing and routing protocols of Chord, we can define a mapping of objects to its corresponding peers in the following manner:

1. Each object x is attached a keyword set $D(x)$ by its content holder.
2. For each $d_i \in D(x)$, object x is mapped to n_i if n_i is a successor of hashed d_i.

For example, the given keywords d_1, d_2, d_3 are assigned to an object f_0, and peers of d_1, d_2, d_3 are n_1, n_2, n_3, respectively. The index entry of f_0 will be mapped to these three peers. This ensures that the indices of objects are approximately[1] and evenly distributed over the participating peers. An example of an index entry is given in Table 1.

Table 1. Structure of an index entry

Key	Metadata	Location
hash(d_l)	album, artist, year	*PeerID:* n_l

As shown in Table 1, an index entry consists of three main fields, i.e., key, metadata and location. The metadata field is used to help users choosing objects to download, as well as supporting metadata search in the system. Here, it is worth noting that an index is a set of entries associated to a given keyword, and an index replica is a copy of the whole entries contained in the index.

Conducting a search for a single keyword is to reveal all of the index entries containing the keyword. The search key is mapped to a peer using Chord. The index for the computed key will be delivered to the requester. To answer a Boolean 'AND' or 'OR' query that contains multiple keywords it fetches the indices of each keyword and calculates the intersection or union of the obtained indices as the final result. In the following section, we discuss the basic keyword search model as described above. But, it is worth noting that by carefully caching the search result of frequently requested keyword sets and parsing queries into proper phases, the performance of the keyword search could be improved.

3.2 Skewed Query Model

Keyword popularity is defined as the fraction of queries associated with the keyword. Suppose that the system receives a query consisting of several keywords. The possibility of the appearance of a keyword in the query can be estimated by two factors: the probability of the query length and the probability of associating each keyword to the query.

In the model, the length of each query is assumed to follow a binomial distribution, as is roughly verified by using monthly statistical data [20]. More concretely, we assume that each query is constructed by using a Bernoulli trial with stopping probability p. For example, the probability that a given query has length i is given as:

$$L_i \overset{def}{=} (1-p)^{i-1} \times p. \tag{1}$$

Let Q denote the set of queries, and L be the maximum query length. Then, according to the basic keyword search model described in Section 3.1, the number of sub-queries generated in the system is:

[1]Consistent hashing produces a bound of $O(\log n)$ imbalance degree of keys between peers, where n is the number of peers in the system.

$$|Q| \times \sum_{1 \le i \le L} (1-p)^{i-1} \times p \times i. \tag{2}$$

Let p_j denote the probability of associating keyword d_j to a query, and we assume that it follows the Zipf's first law. For example:

$$p_j = \frac{1}{(i^s \times H_{|D|,s})}, \tag{3}$$

Where

$$H_{|D|,s} = \sum_{1 \le j \le |D|} (1/i^s). \tag{4}$$

and s is a parameter called the Zipf-parameter. Note that $\sum_{1 \le j \le |D|} p_j = 1$ by definition, where D is the super-set of keywords in the system.

It should be worth noting that under such natural and reasonable assumptions, a number of sub-queries will be generated for getting indices matched to the key of sub-queries. The number of sub-queries associated with each keyword will be highly imbalanced (i.e., a popular keyword is contained in a large number of queries, while an unpopular keyword is rarely contained in the queries).

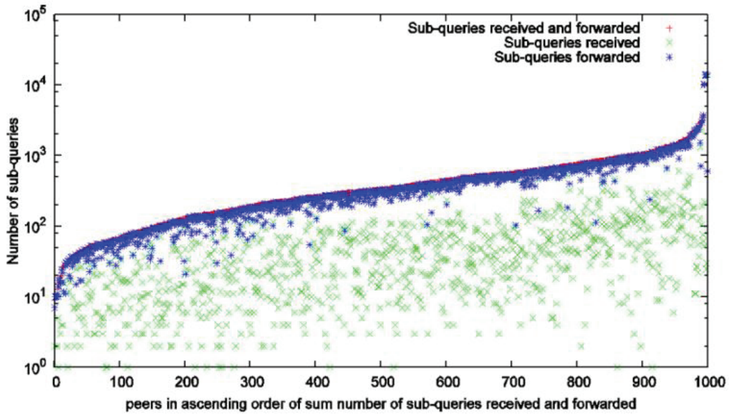

Fig. 1. Sub-queries received and forwarded at each peer.

To better understand this problem, we have performed a simulation to gather the sub-query information of peers in the system. At each peer, we keep track of the number of sub-queries received by the local index, as well as the number of sub-queries forwarded to other peers. In the simulation, we fix N=1,000, and $|D|$=10,000. The total number of queries are 20,000 with Zipf's parameter of s=0.8. The query length parameter is p=0.6 and L=10. A randomly selected peer in the system initiates each query. Fig. 1 shows the result of the sub-queries received and forwarded at each peer. In this figure, it is found that 1) a large number of peers in the system forwarded much more sub-queries than they received; 2) some peers received a large number of sub-queries, but they forwarded few sub-queries; 3) some peers received many sub-queries and also forwarded many sub-queries. These observations justify the importance of routing the load reduction and query load balancing in realizing low latency in index retrieval.

4. PROPOSED SCHEMES

In this section, we first introduce some notations that are used in the paper. Afterwards, we propose three concrete replication schemes for balancing the query load. For example, 1) an active index replication scheme to highly decrease the routing load in the system and to share the response load received by popular indices with replica peers; 2) a proactive pointer replication scheme to reduce the number of index replicas generated by the first scheme; and 3) a passive index replication scheme to guarantee the maximum query load of peers.

4.1 Preliminaries

Remember that $D=\{d_1, d_2, ..., d_{|D|}\}$ is the superset of keywords in the system. In the rest of this paper, d_i is also used for representing a sub-query contained keyword d_i (i.e., sub-query d_i) or an index associated with keyword d_i (i.e., index d_i). Let $w(d_i, n_j)$ denote the number of sub-queries received by peer n_j that stores index d_i (i.e., response load), and let $v(d_i, n_j)$ denote the number of sub-queries initiated or forwarded by peer n_j for index d_i (i.e., routing load). $w(d_i, n_j)$ and $v(d_i, n_j)$ are periodically calculated by every peer during a unit time interval of T. The exponential moving average (EMA) technique is employed to predict the traffic in the next time period. The formula for calculating $w(d_i, n_j)$ and $v(d_i, n_j)$ of peer n_j at time periods $t \geq 2$ is:

$$w^t(d_i,n_j) = \beta w^{t-1}(d_i,n_j) + (1-\beta) w^t(d_i,n_j),$$

$$v^t(d_i,n_j) = \beta v^{t-1}(d_i,n_j) + (1-\beta) v^t(d_i,n_j). \qquad (5)$$

where $\beta \in [0, 1]$. Note that a smaller β makes the new observations relatively more important than a larger β, while a larger β can alleviate replica fluctuation by a sudden and short traffic variation time. An appropriate value of β can be determined according to the actual situation of the system.

Let $\hat{D}$ be the set of indices and be represented as a set of keywords assigned to n_j, and let $\bar{D}$ be the set of sub-queries and be represented as a set of keywords initiated or routed by n_j. The load of n_j, denoted as $\ell(n_j)$, is the weighted sum of its response load and routing load:

$$\ell(n_j) \overset{def}{=} a \sum_{1 \leq j \leq |\hat{D}|} w^t(d_i,n_j) + a \sum_{1 \leq j \leq |\bar{D}|} v^t(d_i,n_j). \qquad (6)$$

In the rest of this paper, we assume that the weighted factor is: a=b=1 (i.e., that the entire routing load is higher than the entire response load in the system). For any given peer n_i, we also define the threshold value, T_0, which is referred to as the target load. The target load of peers is typically defined as follows:

$$T_0 \overset{def}{=} \sum_{n_j \in N} \ell(n_j)/N + c. \tag{7}$$

where $c > 0$ is an appropriate slack.

4.2 Active Index Replication

Under the skewed query model, if we could make a substantial reduction on the search hops for popular indices, the routing load in the system would be highly reduced, and the response load of home peers that stored popular indices will be shared with replica index peers.

The key idea of the first scheme is to bind the maximum response load of each index by an appropriate constant of μ. With this constraint, an index with a popular keyword needs to have more replicas to distribute the response load of the index.

Suppose that peer n_i has an index d_i, then a bottom-up directed weighted tree graph rooted at n_i can be utilized for describing the query flow of searching for d_i. The search by a peer in the tree is the process of moving sub-query d_i along the path towards the root. The level of a peer in the tree is the path length from this peer to the root, where the level of root is 0. In the tree graph, each peer maintains the number of incoming queries for index d_i from its child peers.

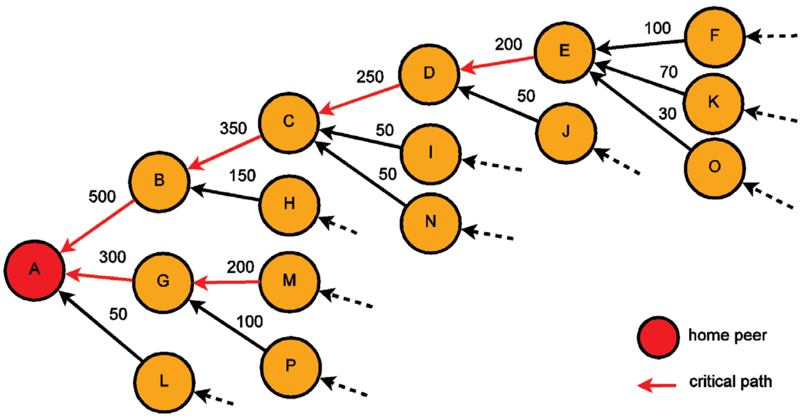

Fig. 2. Query flow in a tree-based routing distributed hash table.

Load information aggregation is conducted using this tree. In order to ensure utilization of replicas, we exclude some peers with low traffic rate from being categorized as candidate peers of index replicas by setting an appropriate minimum load threshold $T_{min} = \alpha\mu$, where $\alpha \in [0, 1]$. Each peer n_x in the tree forwards load information of a child peer ($v(d_i, n_j)$, TTR) to its parent peer, if and only if the child peer n_j satisfies two condition of $v(d_i, n_j) \geq T_{min}$ and its routing load

$v(d_i, n_j)$ is the biggest among n_x's child peers. *TTR* is initialized to 1 and is used for recording the level of peer n_j. In this way, the home peer n_i will maintain R disjoint paths, which is referred to as the critical path of the home peer. Let N_i be the direct child peer set of home peer n_i, then the number of critical paths will be no more than the size of home peer's child peers (i.e., $|R| \leq |N_i|$). An example of a query flow for an index is given in Fig. 2. In Fig. 2, when T_{min}=200, peer A maintains two critical paths (i.e., [B,C, D, E] and [G, M]).

The concrete process of index replication initiated at the home peer or index replica peer that stored d_i is given in Algorithm 1. As shown in Algorithm 1, index replication is triggered when the response load received by d_i at peer n_j exceeds the threshold of μ (line 1). This operation is periodically executed by peer n_j. Each time the threshold is reached, a critical path will be selected if the direct child peer of home peer n_i has the highest routing load for index d_i (lines 3-5). Upon determining the critical path, a utility value of placing replica on each peer in the critical path is calculated, where the utility value means the reduction of the routing load if a replica is placed on that peer. The peer corresponding to the highest utility value will be selected to receive a replica of index d_i (line 6-7). The above process will be repeatedly executed until the response load $w(d_i, n_j)$ is no more than μ (2-8).

Replica deletion is a reverse operation of replica placement and is effective in reducing the maintenance cost of replicas. We are proposing a lightweight replica deletion algorithm that is executed by each replica peer. In replica placement, each replica is attached to a *TTL* field, where *TTL* is the retention period of the index replica. *TTL* can be divided into a number of unit time intervals by T (remember that T is the unit time interval of calculating the traffic rate). Let's denote $w(d_i, n_j) < \delta T_{min}$ ($\delta < 1$), where δ is a under-loaded factor. This indicates the replica as being an infrequently used replica. By checking the last h unit time intervals of *TTL*, if the $w(d_i, n_j) < \delta T_{min}$ condition continuously occurs during h time intervals, the index replica will be automatically deleted after *TTL* is exhausted. Otherwise, *TTL* will be automatically reset to the initial value and consequently, the index replica will be kept for a new retention period. The most important thing to remember is that the determination of keeping index replicas can be executed by each replica peer without extra communication cost.

Algorithm 1. Pseudocode of active index replication of d_i, executed at home peer or replica peer n_i

```
Ensure: $w(d_i, n_i) \leq \mu$
1: if $w(d_i, n_i) > \mu$ then
2:     repeat
3:         for all path $r_i \in R$ do
4:         Choose the path $r_i$ satisfying that $r_i \leftarrow \arg\max_{n_k \in N_i}(n_k, r_i)$.
5:         end for
6:         $d_i, n_j \leftarrow \arg\max_{n_j \in r_i} TTR \times v(d_i, n_j)$.
7:         send a replica of indices associated with $d_i$ to $n_j$.
8:     until $w(d_i, n_i) \leq \mu$
9: end if
```

4.3 Proactive Pointer Replication

The second scheme places a number of pointers for each key (i.e., hashed keyword), in order to reduce the number of index replicas generated in the first scheme. Note that the pointer is a pair of one keyword and its home peer.

Intuitively, each keyword d_i is assigned to a predetermined number of identified points in an ID space. The corresponding peers of these identified points are referred to as 'gateway' peers. The gateway peers divide the ID space into approximately equivalent sub-ranges, each of which maintains a pointer to the home peer of d_i. By having all queries initiated by peers from a sub-range passing through the gateway peer in the sub-range, the height of the query flow tree for the searching index of d_i can be compressed.

A gateway peer can monitor the traffic coming from its sub-range and can then help the home peer with placing more replicas to the high traffic of sub-ranges, which results in effectively reducing the routing load in the system. Moreover, increased disjoint paths towards home peers could further balance the query load of peers.

More concretely, the idea of the second scheme is realized by introducing the system parameter b. By changing the b-bits prefix of the hashed d_i, $2b-1$ identified points are determined. The home peer of d_i places pointers on the corresponding peers (i.e., gateway peer) of these identified points. Then, in the query routing, each requester selects the closest gateway peer based on the ID space distance between the search key and the requester peer. An example of query routing using pointers based on Chord is shown in Fig. 3. As shown in Fig. 3, the hash value of 'key0' is 00xx. In the case of b=2, three gateway peers corresponding to 01xx, 10xx, and 11xx are fixed. A requester in the range between 10xx and 11xx could calculate the distance between itself and 00xx, 01xx, 10xx, and 11xx, respectively. Then, it will choose the closest gateway peer in the clockwise direction (i.e., the peer corresponding to 11xx). Finally, the gateway peer will forward the query to the home peer using the information in the pointer replica.

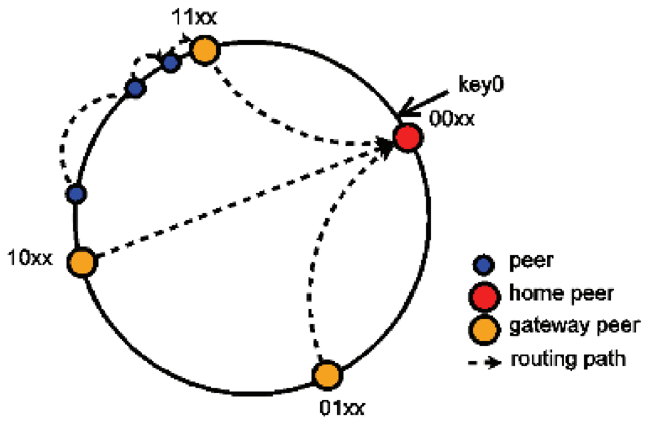

Fig. 3. Query routing after applying pointer replication.

The concrete process of placing pointer replicas based on Chord is given in Algorithm 2. First, let's prepare a null set K and let m-bit k^* be the hash value of keyword d_i (line 1). Then, peer n_i changes the b-bits prefix of k^* using the bit operation function ShiftRight and ShiftLeft and then move the 2b keys into K (line 2-6). Lastly, n_i will send a pointer replica to every corresponding peer that is the successor of $k \in K$ except for n_i (line 7-9).

Algorithm 2. Pseudocode of proactive pointer replication for d_i, executed by home peer n_i

```
1: Let $K = \emptyset$. Let $k^* = hash(d_i)$ and $m$ be the number of bits of $k^*$.
2: Preserve low $(m-b)$ bits of $k$, saved as $l$.
3: ShiftRight$(m-b, k)$, saved as $h$.
4: for all $i \in [0, 2^b]$ do
5: ShiftLeft$(m-b, h \oplus i) + l$, and move it to $K$.
6: end for
7: for all $k \in K \backslash k^*$ do
8: Home peer $n_i$ send a pointer replica to the peer corresponding to $k$.
9: end for
```

4.4 Passive Index Replication

In our system, a peer would be still overloaded due to the summation of the response load and routing load. However, the imbalance of the query load of peers is alleviated after Scheme 1 and Scheme 2 are applied. In order to solve this problem, we distinguished the overloading of peers that mainly result from the routing load and from the respond load. We also effectively determined the operation of pushing or pulling index replicas under each case.

The concrete process of load shedding is given in Algorithm 3. A query counter for counting total number of incoming queries is used at every peer (line 1). If peer n_i exceeds the threshold of query load T_0 (line 2), then the following process is used to justify the main cause of this overloading (line 4-11). If this overloading is mainly caused by the response load (line 4), n_i will choose an index replica of d_i for its neighbor peer n_j, where d_i and n_j are selected by sorting $w(d_i, n_j)$, which is maintained by peer n_i (line 5-6). Otherwise, n_i will pull an index replica from the home peer that stored index d_i, and then push the index replica to its neighbor peer n_j, where d_i and n_j are selected by sorting $v(d_i, n_j)$, which is maintained by peer n_i (line 8-10). This process will be repeatedly executed until query load $\ell(n_i)$ is no more than T_0 (3-12).

Algorithm 3. Pseudocode of excessive load shedding, after the query load of peer n_i exceeds T_0

```
Ensure: $\ell(n_i) \leq T_0$
 1: $query\_counter \leftarrow query\_counter+1$
 2: if $query\_counter > T_0$ then
 3:   repeat
 4:     if $\sum_{1 \leq i \leq |\hat{D}|} w(d_i, n_j) \geq \sum_{1 \leq i \leq |\bar{D}|} v(d_i, n_j)$ then
 5:       $d_i, n_j \leftarrow \arg\max_{d_i \in \hat{D}, n_j \in N_i} w(d_i, n_j)$
 6:       push a replica of indices associated with $d_i$ to neighbor peer $n_j$
 7:     else
 8:       $d_i, n_j \leftarrow \arg\max_{d_i \in \bar{D}, n_j \in N_i} v(d_i, n_j)$
 9:       pull a replica of indices associated with $d_i$ from home peer
10:       push the replica of indices associated with $d_i$ to neighbor peer $n_j$
11:     end if
12:   until $\ell(n_i) \leq T_0$
13: end if
```

The purpose of our third scheme is to eliminate a bottleneck situation from occurring in the system's query processing. Thus, an appropriate target load T_0 should be determined according to the actual situation of the system. It is worth noting that there is a trade-off between the target load and index replication cost incurred by the third scheme. If the target load is set with a high slack value c, the maximum query load of the peers would still be high, and consequently, these peers would create a bottleneck in the system's query processing. On the other hand, if the target load were set with a small slack value c, a large number of index replicas would be created. As such, the maintenance cost of these index replicas in the system would be high.

5. SIMULATION

We evaluated our proposed schemes based on Overlay Weaver [21]. We used Chord as the routing protocol in our simulation. Chord assigns each peer a 160-bit identifier using a SHA-1 hash function. Each peer maintains its successor, predecessor, and finger table with 160 entries. Note that the size of finger table would be incomplete when the system stabilization process of Chord is not conducted enough times. In our simulations, the stabilization period is set from 2 seconds to 128 seconds. After all of the peers join the system the system waits for 1 hour to allow peers to fix their finger tables before uploading the index and conducting query searches.

The simulations in this section were conducted to verify the effect of index replications. Thus, we assumed that each index size was equivalent (e.g. 200 kB), and we measured the cost of index replication in terms of the number of index replicas in the system. In other words, we did not simulate the actual maintenance cost of index replicas, and we did not simulate the size of each object. Finally, the cost of pointer replication has been ignored in the simulation. This is because the size of the pointer replica was small enough.

5.1 Simulation Model

In the following simulations, we fix N=1000, and $|D|$=50, i.e., it is a static setting that the number of peers and the number of keywords will not be changed in simulations. The probability of associating keyword set from D to a query follows the Zipf's distribution with parameter s = 0.8. The total number of queries is fixed to 20,000. The length of each query is determined by a Bernoulli trial with stopping probability 0.4, i.e., the probability of assigning i tags is given by $(0.6)^{i-1} \times 0.4$, while the maximum length of each query is bounded by 10. The time interval of generating queries is 20 ms. The query initiator in each query interval is chosen from peers in the system, and such selection follows a bounded Pareto distribution with parameter 2. The unit time interval T = 1 second, and the EMA parameter β = 0.5. The system parameter b = 4. Note that the weight of response load and routing load are fixed to 1. Finally, we did not set any restriction on the storage capacity of each peer.

5.2 Effect of Replication Schemes

We first evaluate the performance of three replication-based query load balancing schemes. Fig. 4 provides a summary of the results. The vertical axis is the query load of each peer and the horizontal axis is peers arranged in an ascending order of the query load. The four curves in the figure show the initial query load distribution and the query load distributions after the three schemes were applied, respectively. Empirically, we set μ=500 and T_{min}=80 in Scheme 1. The reason for determining T_{min}=80 can be explained using Table 2. In Table 2, we investigated the effect of T_{min} by varying T_{min} from 30 to 180 with an interval of 50. We found that when T_{min} is smaller than 80, it cannot significantly reduce the replication cost. Whereas, a smaller T_{min} means that more load aggregation messages are needed in the system. Thus, we set T_{min}=80 in Scheme 1. In addition, the target query load was T_0=1,000.

As shown in Fig. 4, by applying Scheme 1, the query load (more exactly, routing load) of the system was significantly reduced by 47.26%, as compared to the initial state. Meanwhile, the load variation was significantly reduced from 1,055.79 to 627.52. Furthermore, by combining Scheme 1 and Scheme 2, the query load was reduced by 68.28%, as compared to the initial state, and the load variation was reduced to 262.44. In Fig. 4, it is easy to see that the query load of peers is more balanced after Scheme 2 was applied. Finally, when Scheme 3 was applied, the query load was reduced by 73.95%, as compared to the initial state, and the load variation was further reduced to 165.69.

Table 2. Effect of parameter T_{min}

T_{min}	Number of index replicas
180	175
130	158
80	135
30	134

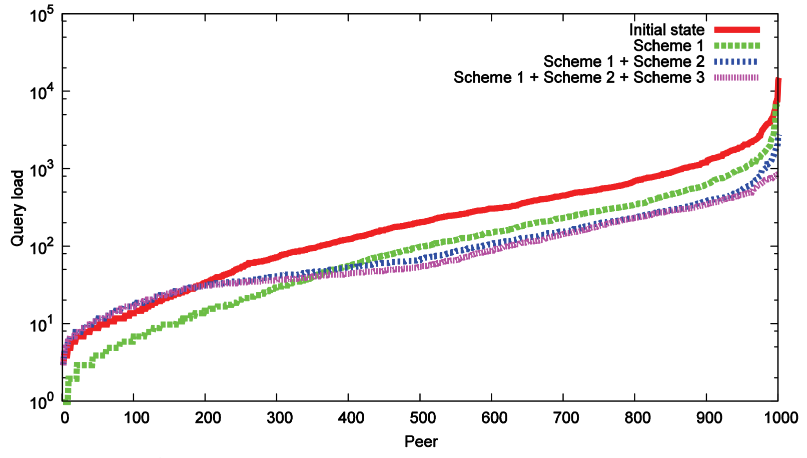

Fig. 4. Load distribution after the replication schemes were applied.

Table 3 shows the cost of replication schemes in terms of the number of index replicas. This result indicates that by applying Scheme 2, it can reduce the number of index replicas generated by Scheme 1. Moreover, by applying Scheme 3, query load of all peers are smaller than target load.

Consequently, we were able to attain a sufficient query load reduction with Scheme 1. After applying Scheme 2, we were able to get more even load distributions, as compared to Scheme 1, and at a smaller cost. There are a few overloaded peers that were not helped by Scheme 1 and Scheme 2, but Scheme 3 effectively resolved them at an acceptable cost.

Table 3. Cost of replication

Scheme	Number of index replicas
Initial State	NULL
Scheme 1	135
Scheme 1 and 2	98
Scheme 1, 2 and 3	120

5.3 Comparison with PCache Replication

In this section, we compare our replica placement schemes with PCache replication. To be fairly comparable, we evaluate the effect of query load distribution when the given number of replicas is equal to 80. Fig. 5 illustrates the results.

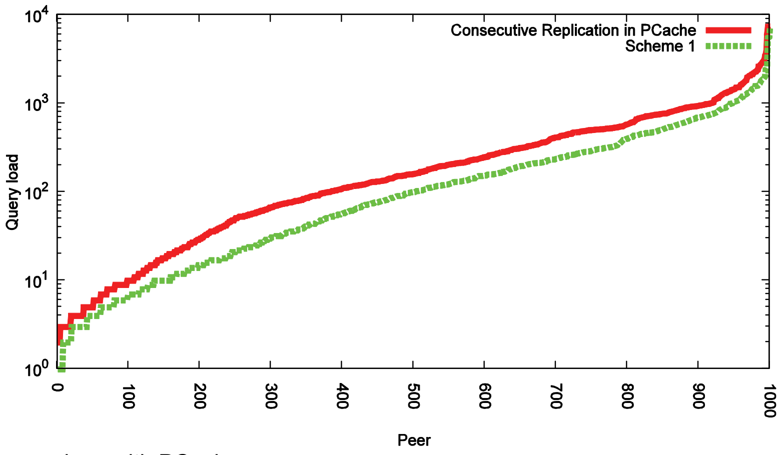

Fig. 5. Comparison with PCache.

As shown in Fig. 5, Scheme 1 can further reduce 34.36% of the query load, as compared to PCache. The load variation of PCache is 713.69, which is higher than the 513.25 in Scheme 1. This is because in each index replication operation, PCache consecutively places the index replicas at the predecessor peers without considering the actual query rate of the replica peers.

5.4 Comparison with EAD

In this section, we compare our replica placement schemes with EAD replication. Under a given target load T_0, we evaluated the index replication cost in terms of the number of index replicas. Fig. 6 illustrates the index replication cost of different schemes. By varying the threshold of response load μ from 500 to 900, we found that the number of index replicas generated by EAD is higher than our schemes. It is worth noting that when μ is smaller, the cost of EAD becomes significantly higher than our schemes. This is because that EAD selects candidate replica peers only according to query rate. This results in a large number of replicas being placed near the home peer. In contrast, in Scheme 1, we used *TTR* to record the level of the candidate replica peers. This allowed us to select candidate replica peers in order to more effectively reduce the routing load than EAD allows.

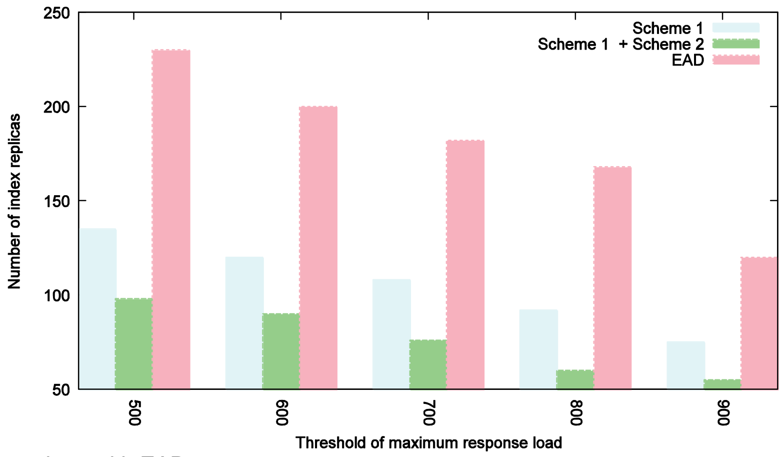

Fig. 6. Comparison with EAD.

6. Concluding Marks

In this paper, we studied the query load balancing problem in DHT-based P2P file search, and proposed an integrated solution that consists of three replication schemes to solve it. We evaluated the performance of the proposed schemes by simulation. The results demonstrate that all of the schemes can work in coordination to avoid the overloading of peers and to alleviate the query load imbalance for the systems. Moreover, by comparing the two prominent replica placement schemes, the cost-effectiveness of the proposed schemes was proved.

However, in practice, P2P systems present many problems, such as low bandwidth or low connectivity peers. Low bandwidth peers impose an effect on performing query processing tasks. On the other hand, low connectivity peers introduce a big challenge in managing replicas. This is due to the fact that these peers usually connect to the system to download a few files and then disconnect. In our future work, we would like to measure the performance of our replication schemes in a more realistic P2P environment. This includes looking at areas such as large size of keyword set, and heterogeneous bandwidth of peers.

References

[1] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: a scalable peer-to-peer lookup service for Internet applications," in *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM2001)*, San Diego, CA, 2001, pp. 149-160.

[2] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," in *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and*

Protocols for Computer Communications (SIGCOMM2001), San Diego, CA, 2001, pp. 161-172.

[3] A. Rowstron and P. Druschel, "Pastry: scalable, decentralized object location, and routing for large-scale peer-to-peer systems," in *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms (Middleware2001)*, Heidelberg, Germany, 2001, pp. 329-350.

[4] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph, "Tapestry: an infrastructure for fault-tolerant wide-area location and routing," *Technical Report*, University of California at Berkeley, CA, 2001.

[5] M. E. Newman, "Power laws, Pareto distributions and Zipf's law," *Contemporary Physics*, vol. 46, no. 5, pp. 323-351, 2005.

[6] V. Ramasubramanian and E. G. Sirer, "Beehive: O(1) lookup performance for power-law query distributions in peer-to-peer overlays," in *Proceedings of the 1st Symposium on Networked Systems Design and Implementation (NSDI2004)*, Berkeley, CA, 2004, pp. 8-8.

[7] W. Rao, L. Chen, A. W. C. Fu, and G. Wang, "Optimal resource placement in structured peer-to-peer networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, no. 7, pp. 1011-1026, 2010.

[8] H. Shen, "An efficient and adaptive decentralized file replication algorithm in P2P file sharing systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, no. 6, pp. 827-840, 2010.

[9] J. Zhou, X. Zhang, L. Bhuyan, and B. Liu, "Clustered K-center: effective replica placement in peer-to-peer systems," in *Proceedings of the Global Communications Conference (GLOBECOM2007)*, Washington, DC, 2007, pp. 2008-2013.

[10] A. Ghodsi, L. O. Alima, and S. Haridi, "Symmetric replication for structured peer-to-peer systems," in *Databases, Information Systems, and Peer-to-peer Computing*. Heidelberg: Springer, 2005, pp. 74-85.

[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 System Principles (SOSP2001)*, Banff, Canada, 2001, pp. 202-215.

[12] B. Godfrey, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica, "Load balancing in dynamic structured P2P systems," in *Proceedings of 23rd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2004)*, Hong Kong, Chana, 2004, pp. 2253-2262.

[13] A. Rao, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica, "Load balancing in structured P2P systems," in *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS2003)*, Berkeley, CA, 2003, pp. 68-79.

[14] S. Rieche, L. Petrak, and K. Wehrle, "A thermal-dissipation-based approach for balancing data load in distributed hash tables," in *Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks (LCN2004)*, Tampa, FL, 2004, pp. 15-23.

[15] D. R. Karger and M. Ruhl, "Simple efficient load balancing algorithms for peer-to-peer systems," in *Proceedings of the 16th Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA2004)*, Barcelona, Spain, 2004, pp. 36-43.

[16] J. Byers, J. Considine, and M. Mitzenmacher, "Simple load balancing for DHTs," in *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS2003)*, Berkeley, CA, 2003, pp. 80-87.

[17] S. Bianchi, S. Serbu, P. Felber, and P. Kropf, "Adaptive load balancing for DHT lookups," in *Proceedings of 15th International Conference on Computer Communications and Networks (ICCCN200)*, Arlington, VA, 2006, pp. 411-418.

[18] H. Shen and C. Z. Xu, "Elastic routing table with provable performance for congestion control in DHT networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, no. 2, pp. 242-256, 2010.

[19] C. Harvesf and D. M. Blough, "Replica placement for route diversity in tree-based routing distributed hash tables," *IEEE Transactions on Dependable and Secure Computing*, vol. 8, no. 3, pp. 419-433, 2011.

[20] Trellian, "Keyword and search engines statistics," http://www.keyworddiscovery.com/keyword-stats.html.

[21] K. Shudo, Y. Tanaka, and S. Sekiguchi, "Overlay Weaver: an overlay construction toolkit," *Computer Communications*, vol. 31, no. 2, pp. 402-412, 2008.

Qi Cao

He received B.E. degree from Tianjin University of Science and Technology in 2007 and M.E. degree from Hiroshima University in 2010. He is a Ph.D. student at Department of information engineering, Hiroshima University. His research interests are in the field of design, implementation and performance evaluation of peer-to-peer systems.

Satoshi Fujita

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