A New Caching Technique to Support Conjunctive Queries in P2P DHT

Koji KOBATAKE†, Nonmember, Shigeaki TAGASHIRA*, and Satoshi FUJITA†, Members

SUMMARY P2P DHT (Peer-to-Peer Distributed Hash Table) is one of typical techniques for realizing an efficient management of shared resources distributed over a network and a keyword search over such networks in a fully distributed manner. In this paper, we propose a new method for supporting conjunctive queries in P2P DHT. The basic idea of the proposed technique is to share a global information on past trials by conducting a local caching of search results for conjunctive queries and by registering the fact to the global DHT. Such a result caching is expected to significantly reduce the amount of transmitted data compared with conventional schemes. The effect of the proposed method is experimentally evaluated by simulation. The result of experiments indicates that by using the proposed method, the amount of returned data is reduced by 60% compared with conventional P2P DHT which does not support conjunctive queries.

key words: P2P DHT, conjunctive query, result cache

1. Introduction

An explosive spread of the information technology rapidly increases the number of users who acquire various information and resources distributed over the network. Such an increase of the number of users significantly enlarges data flows over the network, which severely concentrates the load of specific servers under conventional client/server (C/S) systems, with respect to the data storage to store the contents of shared objects and the CPU time for processing queries issued by the clients. As a means to overcome such serious situations, peer-to-peer systems (P2P) attention in recent years [1], [8].

P2P is a distributed system consisting of several computers called nodes or peers, which can communicate with each other through an underlying interconnection network. In P2P, contents of objects are distributed over the network, and managed by each peer participating in the system in a distributed manner. Hence, before accessing an object in P2P, each peer must learn about the name of peers who manages the contents of the object. Previous schemes to realize such an indexing to the objects could be classified into the following three types: hybrid type, flooding type, and P2P distributed hash table (P2P DHT).

In the hybrid type, all indexes are maintained by a centralized server called index server. Thus, each peer can easily acquire the index of the requested object by simply referring to the index server [10], while it does not resolve the problem of overloading of specific servers in C/S systems. In the flooding type, the search of an object is realized by flooding a query message over an overlay network; i.e., upon receiving a query from an adjacent peer in the overlay, it notifies the search result to the originator of the query if it holds a matching object, and otherwise, it forwards a copy of the query to the other adjacent peers. The number of hops taken by a query is generally bounded by an appropriate value in order to prevent an unnecessary increase of the message traffic. Examples of the flooding type P2P include Gnutella [7] and Freenet [5]. In contrast to the above two types, in P2P DHT, indexes to objects are stored in a logical space constructed over an overlay network, and the forwarding of query is systematically controlled to prevent an increase in the message traffic (a detailed explanation of P2P DHT will be given in the next section). Examples of P2P DHT include Chord [14], CAN (Content-Addressable Network) [11], Pastry [13], and Tapestry [15].

As will be described later, in P2P DHT, objects managed by the system are identified by an identifier (ID) given to the objects. Thus, we could surely obtain the contents of an object if we accurately specified the ID of the object. Although an ID may be the file name of the object, we can associate several keywords to each object as ID’s, and use them to realize a keyword search (in the literature, such method is referred to as the reversed DHT [12]), and in the following, we will focus our attention to P2P DHT supporting keyword search which is one of general applications for P2P DHTs. The main goal of our research is to realize an advanced keyword search engine for distributed resources (e.g., WWW resources) in a P2P fashion. A critical problem in realizing keyword search in an efficient manner is how to bound the number of objects matching the given keyword. The number of matching objects seriously increases as the number of peers participating in the system grows, which will make it difficult to find target objects that we’re looking for. This problem could be (partially) solved by executing a conditional search such as the processing of conjunctive queries consisting of several keywords because the search result can be narrowed down by specifying multiple keywords. The technique for supporting such conjunctive queries is indispensable in keyword search systems and it has been normally supported in existing WWW search engines. Unfortunately however, most conventional P2P DHT approaches do not support conjunctive (or con-
ditional) queries consisting of several keywords, and the result for such queries could be obtained by merely integrating the search results for all keywords, which causes many request messages to the P2P DHT and large transmitted data for search results on processing a conjunctive query.

In this paper, we propose a new technique for supporting conjunctive queries in P2P DHT. The main goal of the proposed scheme is to efficiently process conjunctive queries in such distributed systems (i.e., without centralized servers). The basic idea of the proposed method is to cache the search result for conjunctive queries on the local memory of each peer, and to register the information on the cached conjunction to the DHT. In the literature, this kind of cache is referred to as the result cache [2], [12], and it is recognized as a general technique to increase the efficiency of database search. The proposed method appropriately controls the set of cached conjunctions to increase the efficiency of memory utilization, and introduces a navigation mechanism to urge the sharing of cached information. The effect of the proposed method was experimentally evaluated by simulation. The result of experiments indicates that although the use of result cache reduces the amount of returned data by 60%, it increases the number of meaningless messages compared with conventional methods.

The remainder of this paper is organized as follows. Section 2 describes a model of keyword search in P2P DHT. Section 3 describes the proposed method, and the evaluation of the method is given in Sect. 4. Related work on the keyword search in P2P DHT is outlined in Sect. 5. Finally, Sect. 6 concludes the paper with future problems.

2. Model

This section describes the model considered in this paper. In the following, we describe the model of P2P, details of P2P DHT, and the model for keyword search in P2P DHT.

2.1 P2P

In the following, we consider a situation in which each peer holds contents of several objects, and those objects are shared by all peers participating in the system. Each object is associated with several keywords, which could be regarded as an identifier (ID) of the object. An index of an object is a pair of an ID of the object and a value that represents the name and IP address of the peer holding the contents of the object (since there may exist several contents associated with the same ID, index is not a function but a relation, and it is implemented as a list of pairs in practice). Thus, if we could have a table of such indexes beforehand, we can efficiently find the name and the address of the peer holding the desired contents by simply referring to the index table with a given ID. In fact, P2P DHT described below is a means to realize such an index table on P2P in a distributed manner.

2.2 P2P DHT

P2P DHT is an index management method based on the notion of distributed hash tables (DHT), where "hash table" refers to a logical space for storing (lists of) indexes. Assignment of index to a point in the hash table is conducted by using an appropriate hash function from the set of ID's to the hash table. Each peer participating in P2P DHT manages a part of the hash table, in such a way that the collection of those parts covers the entire hash table. Each peer also holds the information of peers which is assigned adjacent part in the hash table to form a (logical) link to the adjacent peers. Thus, message transmission to a peer managing a point in the hash table could be realized by repeating the forwarding of the message to an adjacent peer nearest to the target point, where the forwarded message is eventually received by the peer who manages the point, and will be processed appropriately. In P2P DHT, registration and search of index are defined as primitive operations. The search for an index of object with a designated ID is realized by sending a query message towards the point calculated by the hash function from the ID. Similarly, registration of an index is realized by sending a registration message towards a calculated point in the hash table.

2.3 Retrieval Model

Let S be an (infinite) set of keywords. Let $C(a_i)$ denote the set of contents held in the system which are associated with keyword $a_i \in S$. Recall that in P2P DHT, indexes for all contents in $C(a_i)$ are stored at the point in the hash table that is mapped from keyword $a_i$. In the following, we assume that each query is represented as a conjunction of several keywords, and represent a query consisting of $k$ keywords $a_{i1}, a_{i2}, \ldots, a_{ik}$ as set $Q = \{a_{i1}, a_{i2}, \ldots, a_{ik}\}$ (thus, the number of keywords contained in query $Q$ is given by $|Q|$). Note that query $Q$ requests index information of all contents contained in set

$$C(a_{i1}) \cap C(a_{i2}) \cap \cdots \cap C(a_{ik}) = \bigcap_{a \in Q} C(a).$$

In the following, we assume that keywords contained in a query issued by a peer are selected from set S with a predetermined probability distribution, and that the selection of keywords associated to each object follows the same probability distribution. Note that this assumption considers a situation in which there is a difference of "popularity" among keywords, in such a way that there is an unpopular keyword that is contained in very few number of queries, while there

---


1 The definition of adjacency depends on the definition of the logical space.
is a popular keyword that is contained in almost all queries. In this paper, we use the Zipf’s first law as a concrete model of such probability distribution\(^1\), which is described as follows: when the occurrence frequency of the most popular keyword is \(p\), the occurrence frequency of the \(i\)th popular keyword is represented as \(p/i\). In the following, we assume that keyword \(a_i \in S\) is the \(i\)th popular keyword.

3. Proposed Method

In this section, we propose a new technique to support conjunctive queries in P2P DHT, based on the notion of result cache. In this paper, we refer to a request conjunction issued from a user (or a requester) as “query” and a request message issued from the system as “message”.

3.1 Result Cache

3.1.1 Outline

Result cache is a common technique to avoid meaningless repetitions of messages by temporarily caching search results obtained so far [2], [12]. More concretely, for processing such a conjunctive query in the conventional method, a peer must obtain indexes of all contents associated with each keywords contained in the query; It is necessary to send a message for acquiring indexes of each keyword to the P2P DHT and receive its indexes. The search result can be obtained by calculating a conjunction of indexes for all keywords. For example, let \(Q = \{a, b\}\) be a given query. In the conventional method, the number of transmitted messages is \(|Q|\) and the amount of transmitted indexes is \(|C(a)| + |C(b)|\) for processing the given query, although the size of the search result can be reduced to \(|C(a) \cap C(b)|\). On the other hand, for result caching, the peer caches the search result on its own local memory and registers an index for the cache to the P2P DHT in the same manner which was described in Sect. 2.2; An index of a cache consists of its conjunction (ID), and the name and the address of the peer holding the cache (value). Another peer that requests the same query will find the peer holding the cache by referring to the index using the P2P DHT. It can simply obtain the search result for the query by using the cache.

An advantage of result caching in processing conjunctive queries is as follows: The first one is a reduction of the number of messages issued to the system. For example, although it generally needs three messages to process a query consisting of three keywords, if the result for the whole query has already been cached in the system, we can obtain the same result by simply referring to the cache. Another advantage is that it reduces the amount of returned indexes as increasing the length of conjunctions. For example, although the amount of transmitted indexes to query “\(a \land b\)” is \(|C(a)| + |C(b)|\) in the conventional method, it could be reduced to \(|C(a) \cap C(b)|\) if the result for the query has already been cached in the system.

In this section, we propose a new caching method to support conjunctive queries in P2P DHT. The basic idea of the method is to extend the original DHT in such a way that it keeps the index of peers who cache the result of conjunctive queries. Furthermore, we introduce the following two key techniques to increase the efficiency of the caching scheme:

- We will limit the maximum conjunction length in order to increase the memory utilization ratio.
- We will introduce the notion of index navigator to urge the sharing of cached information.

In the following, we explain the above two points in detail.

3.1.2 Maximum Conjunction Length

In the proposed scheme, we limit the maximum number of keywords contained in the cached conjunction, where the maximum value is referred to as the maximum conjunction length and is denoted by symbol \(L\) in what follows. By setting up the maximum conjunction length appropriately, we could increase the efficiency of memory utilization without reducing the hit ratio to the cache (the effect of such improvement will be experimentally evaluated in Sect. 4).

In the scheme, a conjunctive query consisting of several keywords is processed as follows: If the length of the query is smaller than or equal to \(L\), it should try to acquire the cached result for the whole query, since it should be cached in the system if the same query has already been issued, and the cached result has not been expired. On the other hand, if the length of the given query is greater than \(L\), we have to partition it into small fragments with length at most \(L\), and integrate search results for those fragments since the result for the whole query is not cached on the system. Thus, in the proposed scheme, we will take an approach to cache the (partial) result corresponding to a part of given query if necessary, i.e., the target of caching is not restricted to the result for the whole queries issued by the peers.

3.1.3 Index Navigator

In the proposed scheme, we introduce a navigation mechanism to support such a partition of the given query into small fragments in an appropriate manner, and to support the processing of those partial queries. In this paper, we call this mechanism index navigator. The objective of index navigator is to provide an indicator to show the way of partitioning the given query to obtain a requested answer to the whole query as quickly as possible.

More concretely, we extend the index stored in the hash table in the following manner: First, for each conjunction \(p\) registered in the hash table, we store a list of conjunctions cached on the system that include \(p\) as a subset, in addition to the index about the peer holding the result cache for \(C(p)\).

\(^1\)Note that the difference of such probability distribution does not affect to the basic design of the keyword search scheme. The effect of distribution should be considered in the tuning of parameters used in the keyword search schemes.
In the following, we use $U^*(p)$ to denote the list of such conjunctions. Note that $p \in U^*(p)$. Next, we modify the query processing in such a way that the requester attaches a conjunction $q \supseteq p \subseteq Q$, which is to be examined to obtain the result for the original query $Q$, to the partial query $p$ that is a subset of the original query $Q$ (note that if it has already examined partial conjunction $r$, then it must hold $Q = q \cup r$).

In the proposed scheme, after receiving message $p$ with navigation information $q$, the target peer who manages the index for $p$ returns index information for $C(p)$ to the requester (if any), and in addition, it returns its next navigation information $q'$ that is calculated as $q' \in U^*(p) \setminus p$. Note that when $p$ is a conjunction for a query, $C(p)$ represents a set of caches for the search result of query $p$.

We show a simple example for the index navigation mechanism. A query $Q = \{a, b, c, d, e\}$ is given and search results for conjunctions $\{a, b\}$ and $\{a, b, d\}$ have been already cached on the P2P DHT system. Hence, if a requester sends a request message for conjunction $\{a, b, c\}$ with navigation information $\{d, e\}$, the system returns indexes for $C(\{a, b\})$, and in addition, conjunction $\{a, b, d\}$ as navigation information.

### 3.2 Procedure

In this section, we explain the following two procedures in order: 1) procedure to process conjunctive queries, and 2) procedure to update the result cache. In the following explanation, we use symbol $Q$ to denote the query issued by a user, symbols $p, q$ to denote partial queries (i.e., subset of $Q$) generated in the procedure.

#### 3.2.1 Processing of Conjunctive Query

A peer who issues query $Q$ (i.e., the requester of query $Q$) executes the following procedure:

**procedure SEARCH($Q$)**

1. Let $q \leftarrow Q$.
2. Initialize variable $j$ to $\min(L, |q|)$, where $L$ is the maximum conjunction length.
3. Let $p$ be the prefix of $q$ of length $j$ (i.e., the conjunction of the first $j$ keywords in $q$).
4. To examine if $C(p)$ is cached or not, the peer sends a request message to the system with conjunction $p$ attached with navigation information $q$, and receives an answer to the message. The answer includes indexes for $C(\{a, b\})$ and its next navigation information $q'$, if $C(p)$ is cached. Note that when $p$ is a conjunction for a query, $C(p)$ represents a set of caches for the search result of query $p$.
   - When $C(p)$ is not cached: if $j = 1$, then terminate the procedure since the system holds no objects corresponding to query $Q$. Otherwise, go to Step 3 after reducing $j$ by one.
   - When $C(p)$ is cached: if $q'$ contains no elements other than $p$, then it sends a message to the peer contained in the returned indexes to acquire $C(p)$. Otherwise, after letting $p$ be a longest conjunction contained in $q'$, acquire $C(p)$ in the same manner.
5. Let $q \leftarrow q \setminus p$. If $q = \emptyset$, then take an intersection of subsets acquired so far, and output it as the search result. Otherwise, go to Step 2.

Figure 1 shows an example for procedure SEARCH. A query $Q = \{a, b, c, d, e\}$ is given and the maximum length is set to be three. We assume that search results for several conjunctions have been already cached on the P2P DHT as shown in Fig. 1. First, a requester sends a message for conjunction $\{a, b, c\}$ with navigation information $\{d, e\}$. In this example, there is no entry for conjunction $\{a, b, c\}$ on the P2P DHT. Next, it sends a message for conjunction $\{a, b\}$ with navigation information $\{c, d, e\}$. It receives indexes for $C(\{a, b\})$ attached with conjunction $\{a, b, d\}$. Thus, the requester selects conjunction $\{a, b, d\}$ because it is the longest conjunction. It obtains indexes for $C(\{a, b, d\})$ from the P2P DHT and sends a request message to peer $W$ which is contained in the obtained indexes for acquiring $C(\{a, b, d\})$. Finally, the requester sends a message for conjunction $\{c, e\}(= \{a, b, c, d, e\} \setminus \{a, b, d\})$. There is no entry for $\{c, e\}$. It can obtain indexes for keywords $c$ and $e$ from the P2P DHT.

The termination condition of this procedure is whether it fails the retrieval (i.e., when $j = 1$ and “$C(p)$ is not cached”) or it succeeds the retrieval (i.e., when $q$ becomes empty). Note that in both cases, the requester locally keeps sets of indexes acquired thus far, except for the special case in which the search result for the first keyword in $Q$ is empty. In the following, we name those sets as $C_1, C_2, \ldots, C_l$ in the order of their acquisitions, and let $q_i$ be a conjunction corresponding to set $C_i$. Note that keywords in each $q_i$ is sorted in the lexicographic order. It holds $\bigcup_{i=1}^l q_i = Q$ after terminating the procedure with success. In the following, we use symbol $Q$ to denote the set of those conjunctions, i.e., $Q = \{q_1, q_2, \ldots, q_l\}$.

![Fig. 1 An example for procedure SEARCH.](image)
3.2.2 Update of Result Cache

The update of result cache is conducted immediately before the end of the above mentioned search procedure. In the update procedure, we first partition the set of conjunctions \( Q \) into several subsets \( Q_1, Q_2, \ldots, Q_r \) to satisfy the following two conditions:

- **Condition 1:** For each \( 1 \leq i \leq x, |\bigcup_{q \in Q_i} q| \leq L \).
- **Condition 2:** For each \( 1 \leq i < x, |\bigcup_{q \in Q_i} q| + |q_j| > L \)

where \( q_j \) is the conjunction with the smallest subscript in subset \( Q_{i+1} \).

Note that this partition is realized by executing a greedy bin packing of “items” in \( Q \) to “bins” of size \( L \) each.

Update of the result cache is conducted for each subset \( Q_i \) independently. A concrete update procedure for subset \( Q_i \) is as follows:

1. Let \( Q_i = \{q_{i1}, q_{i2}, \ldots, q_{iy}\} \).
2. Let \( \tilde{q}_j = q_{i1} \cup \cdots \cup q_{ij} \) for each \( 1 \leq j \leq y \) (e.g., \( \tilde{q}_1 = q_{i1} \), \( \tilde{q}_2 = q_{i1} \cup q_{i2} \), \( \tilde{q}_3 = q_{i1} \cup q_{i2} \cup q_{i3} \), and so on).
3. For each \( 1 \leq j \leq y \), calculate \( C(\tilde{q}_j) \), store the result for \( \tilde{q}_j \) on its local cache, and register the fact to the hash table by using conjunction \( \tilde{q}_j \) as the keyword (note that it should register the fact that keywords \( \tilde{q}_2, \tilde{q}_3, \ldots, \tilde{q}_y \) are registered in the hash table as the navigation information, when it registers keyword \( \tilde{q}_1 \) to the hash table).

We show an example for the update procedure in the same situation as Fig. 1. In this situation, \( Q \) is \( \{q_1 = \{a, b, d\}, q_2 = \{c\}, q_3 = \{e\}\} \), \( Q \) can be partitioned into \( Q_1 = \{q_1\} \) and \( Q_2 = \{q_2, q_3\} \) according to the above two conditions. The update procedure for the result cache is conducted for each subset \( Q_1 \) and \( Q_2 \). In subset \( Q_1 \), \( C(\tilde{q}_1(= q_1)) \) is registered to the hash table. In subset \( Q_2 \), \( C(\tilde{q}_2(= q_2)) \) and \( C(\tilde{q}_3(= q_2 \cup q_3)) \) are registered. Note that when \( C(\tilde{q}_1) \) is registered, \( C(\tilde{q}_2) \) is also registered to the same point of the hash table as the navigation information.

4. Evaluation

In this section, we evaluate the effect of the proposed method by simulation. Recall that the objective of the proposed method is to reduce the number of messages and the amount of indexes returned as the search result. In the following, we evaluate the amount of returned indexes in terms of the amount of returned data (Byte) for each query, and the number of messages in terms of the number transmitted messages before receiving the search result for each query. Here, we have to note that in the proposed method, the processing for a given conjunctive query completes when the requester receives the indexes of “objects” matching with the query; i.e., it must send an additional message after receiving the index of “conjunctions” cached on the system (in the following, we do not consider the action to acquire the contents of objects from the received index).

4.1 Simulation Model

The simulation considers two-dimensional CAN as a concrete P2P DHT, with the following assumptions: 1) the set of peers does not change during the simulation, 2) all objects are shared by all peers, and 3) the set of objects does not change during the simulation. Although the simulation under a dynamic P2P environment is one of the most important issues in evaluating the proposed scheme, we provide the simulation under the static environment as a basic evaluation in this paper. We address this issue as an interesting direction of the future work. Peers in CAN can communicate with each other through an underlying communication network, and the time required for the communication between two peers depends on the number of hops on the overlay network and the message length. In the simulation, each peer repeatedly issues queries according to a Poisson distribution with mean \( \lambda = 20 \) (we associate a time unit to 15 sec in the real time; thus 20 time units correspond to 5 min in the real time). The setup for a time unit is not intended to represent a meaningful configuration because we do not show the performance evaluation in terms of the response time in this paper. What is important in the experimental setup is to uniformly issue queries by all peers and store the results on their local caches. Each run of simulation consists of 5,000 queries, i.e., a run of simulation terminates when the total number of queries issued by the peers reaches 5,000.

The cache size of each peer is given as a parameter in the simulation, where the cache size is represented in terms of the number of indexes held in the cache, e.g., “cache size 100” implies that it can hold at most 100 indexes. The contents of cache will be replaced by using an appropriate cache replacement scheme, where in the following, we adopt LRU (Least Recently Used) as the concrete cache replacement scheme. In order to maintain the consistency, we assume that each peer correctly reflects the cached information to the hash table after expiring an index from its local cache, where in the following, we assume that the cost for the maintenance is sufficiently small and negligible.

We assume that set \( S \) contains 2,500 keywords, each peer holds contents of 100 objects, and each object is associated with 20 keywords drawn from \( S \). This configuration enables us to adequately categorize objects shared by all peers. The number of keywords contained in each query is determined by a Bernoulli trial with stopping probability 1/4; i.e., each query consists of two keywords with probability 1/4, three keywords with probability (3/4) \times (1/4) = 3/16, four keywords with probability (3/4) \times (3/4) = 12/16, and so on (to avoid too long queries, we repeat the same process from length two if the number of keywords exceeds ten). Let \( X \) be the number of keywords to be associated with a query (or an object). Then, it selects \( X \) keywords from \( S \) (without replacement) according to the Zipf’s law, and associate them to the query (or the object). This simulation introduces a locality on the selection of queries in addition to the popularity of keywords realized by the Zipf’s law. This locality is
realized by using the notion of working set, which is a set of queries consisting of top 10% of keywords in $S$. This locality model is similar to that employed in [2]. In [2], queries used in the experiments are based on WWW queries and follow the same statistical characteristics as search.com [3] queries using keywords from the TREC-Web data set [4]. Furthermore, the working set model is used as a query locality, where 90% of all queries are issued from the working set. In the following experiments, we fix the size of the working set to 500, and either 30% or 90% of queries issued by the peers are selected from the working set; Note that we evaluate the performance of the proposed scheme under also such low locality (i.e. 30%), in order to understand how sensitive it is to the low degree of locality.

4.2 Effect of Result Cache

In the following, we examine the impact of the maximum conjunction length $L$ and the cache size to the performance of the proposed schemes, in more detail.

4.2.1 Impact of Maximum Conjunction Length

An advantage of increasing maximum conjunction length $L$ is that it enables direct access to the cached result for long queries. On the other hand, it has a disadvantage such that the hit rate to the cache significantly decreases since long queries appear very rarely; i.e., the probability that a long query is issued several times within a short time period is very low. In the proposed scheme, such a decrease of the hit rate increases the number of “meaningless” messages for conjunctions that are “not” cached on the system. Moreover, the number of messages for the update procedure of result cache increases as increasing $L$ especially when a query is long. This is due to the combination for conjunctions of the query to be updated. As was described in Sect. 3.2.2, the size of each subset for $Q$ becomes larger with increase of $L$, although the number of subsets becomes smaller. This causes larger number of messages for registrations to P2P DHT in step (3) of the update procedure. Thus, it is natural to guess that these issues are main reasons of the increase of the number of messages in the proposed scheme.

In order to verify the above conjecture, let NR denote the conventional scheme which does not support conjunctive queries, and RC denote the proposed scheme with result cache (without index navigator). First, we evaluate the performance of those schemes by assuming that each peer has a sufficiently large cache. Figure 2 illustrates the result. The horizontal axis of the figure is the maximum conjunction length $L$, and “$L = 1$” in the figure corresponds to NR. Figure 2(a) indicates that even in such a specific situation, although RC reduces the number of returned indexes by about 80%, the amount of reduction is not affected by the value of $L$. In other words, the amount of reduction is significant only when $L$ is increased from one to two. On the other hand, the number of messages monotonically increases as increasing $L$ as shown in Fig. 2(b), which means that the merit of increasing $L$ is not greater than its demerit.

4.2.2 Impact of Cache Size

Next, we evaluate the impact of the cache size to the performance. Figure 3 shows the result when the locality of issued queries is 30%. Recall that in this simulation, “locality” represents the probability that queries issued by the peers are selected from the working set as was described in Sect. 4.1. Horizontal axis of the figure is the cache size, and each curve in the figure corresponds to the value of $L$; e.g., “level 2” implies “$L = 2$.” From Fig. 3(a), we can observe that the number of returned indexes converges to a specific value by increasing the cache size, while the converged value is (almost) equal for all values of $L$. Thus, in such a case with low locality of query selections (i.e., 30%), we could not reduce the number of returned indexes by increasing the value of $L$ for large cache size, since the number of hits to the cached conjunctions is not high. A similar phenomenon could be observed for the number of messages as shown in Fig. 3(b), where the number of messages monotonically increases as increasing the value of $L$, i.e., the demerit is still greater than

![Fig. 2 Impact of the maximum conjunction length to the performance.](image-url)
We then evaluate the impact by increasing the locality of queries to 90%. Figure 4 illustrates the result. From Fig. 4 (a), we could observe a difference of the number of returned indexes for each $L$, which is greater than Fig. 3, although the difference becomes smaller by increasing $L$. This is because such a high locality improves the hit rate to the cache not only for short queries but also for long queries. Therefore, the increase of $L$ can improve the effectiveness of the direct access to the cached result for long queries under such a high locality. Figure 4 (b) shows that the number of messages firstly increases with increase of $L$, which is similar to Fig. 3 (b). However, the number of messages begins to decrease after $L$ surpasses six under such a high locality. We can explain the reason as follows; In $L = 2$, although the proposed scheme cannot fully gain the effect of the direct access to the caches, it can minimize the cost for the update procedure as was described in Sect. 4.2.1. In contrast, in $L = 10$, it can improve the effectiveness of the direct access. However, it must pay the high cost for the update procedure. In $4 \leq L \leq 8$, the total performance of the proposed scheme is dominated by these demerits, thus it could be degraded as compared to the case of $L = 2$ and $L = 10$.

The above results could be summarized as follows: by using result cache, the number of returned indexes reduces compared with NR, and the amount of reduction increases as increasing $L$. For example, the amount of reduction is 80% when $L = 10$. On the other hand, the number of messages increases as increasing $L$ compared with NR. Thus, in order to attain a sufficient performance in RC, we have to select the value of $L$ appropriately depending on the cache size and the average query length.

4.3 Effect of Index Navigator

Finally, we evaluate the effect of the index navigator. In the following, we refer to the proposed scheme with index navigator as IN. The result is shown in Table 1 (for comparison, the table includes the result for NR, RC, and RC+IN).

As shown in the table, IN reduces the number of returned indexes and the number of messages. It is because of the effect of navigation mechanism which navigates the requester to issue an appropriate partial queries that makes possible to obtain the final search result quickly. In fact, by
using IN, the utilization of cache increases, which reduces the number of returned indexes by 14 to 20%, and the number of messages by 9 to 23%.

5. Related Work

This section surveys conventional techniques to realize efficient keyword search in P2P DHT.

An extension of the hash function to the function from the set of keywords to the hash table was originally proposed by Vahdat [12]. This method is referred to as the reversed DHT method in the literature. In the reversed DHT, each keyword associated with objects are mapped to a point in the hash table by an appropriate hash function, and a list of indexes associated with the keyword is stored at the calculated point. Thus, each user can acquire the index of objects associated with a keyword by applying hash function to the keyword, and by sending a query message to the calculated point. Note that P2P DHT considered in this paper is based on the reversed DHT as was described in Sect. 2.

Bhattacharjee proposed a data structure called view tree to support an efficient processing of conjunctive queries in P2P DHT [2]. View tree realizes a result cache on P2P DHT, and efficiently supports the processing of conjunctive queries by maintaining cached conjunctions in a tree-structured overlay network in a hierarchical manner. Each vertex in the tree keeps the search result for a conjunctive query similar to our proposed method. The tree is maintained in such a way that the parent vertex corresponds to a conjunction that is the prefix of the conjunction corresponding to a child vertex. When a conjunctive query is submitted to a view tree, it first examines if the result for the query is cached in the tree by conducting a tree traversal in a depth-first manner. If it is cached, the requester could obtain the search result by simply refer to the corresponding vertex, and otherwise, the requester inserts a new vertex corresponding to the query to the view tree after obtaining the search result for the query (in a similar manner to the proposed scheme). View tree seems to be a natural realization of the result cache on P2P DHT. However, this method has the following serious drawbacks:

1. It does not provide a means to directly access a vertex corresponding to a given conjunction. Thus, the view tree may contain several candidate vertices for a given conjunction, which causes frequent backtracks during the depth first search.
2. To realize a depth first search, each vertex must keep the information about all children in the view tree. However, it is not practical, since the number of children of a vertex could be as large as the number of all keywords issued by the peers.
3. It does not distinguish popular keywords from others. We need some mechanism to separately deal with such popular keywords, in order to equalize the load of peers.

As a means to resolve the third problem described above, Liu proposed a method called keyword fusion [9]. Keyword fusion is a technique to reduce the load of peers who manage index for popular keywords by appropriately extracting, and by separately dealing with such keywords. More concretely, it first extracts popular keywords by referring to the access history, and registers them to a dictionary called fusion dictionary. It then generates a new keyword for each of unpopular keywords associated with the same object with a popular one, by executing a fusion of popular and unpopular keywords. For example, suppose that object A is associated with three keywords a, b, and c, and that a is a popular keyword. Then, it generates new keywords (a, b) and (a, c) from b and c after registering a to the fusion dictionary. Keywords registered to the fusion dictionary are eliminated from the hash table. Thus, by using this method, although the load of peers could be equalized, it causes another problem such that the keyword search with a single popular keyword becomes impossible.

Gnawali proposed a notion of keyword set to support conjunctive queries [6]. Keyword set is a set of conjunctions that is obtained by integrating keywords associated with each object. By registering conjunctions contained in the keyword set to the system beforehand, we could realize an efficient processing of conjunctive queries. A drawback of this method is that we have to register all conjunctions generated from keywords associated with objects without considering the popularity of keywords and conjunctions, which significantly increases the number of meaningless registrations. A practical solution to this problem is to extract popular keywords from the past history, and to construct keyword set for those popular keywords in an adaptive manner, which is almost the same with the basic idea adopted in the view tree and our proposed method.

6. Concluding Remarks

This paper proposed a new method to support conjunctive queries in P2P DHT. The basic idea of the method is to share global information on past trials by locally caching the search result for conjunctive queries and by registering the fact to the global DHT. The effect of the proposed method is experimentally evaluated by simulation. The result of experiments indicates that by using the proposed method, the amount of returned data is reduced by 60%, compared with conventional P2P DHT.

A future problem is to apply the proposed scheme to a dynamic P2P network in which arrival and departure of peers frequently occur. We have to develop a new technique to keep the hit rate of the cache under such a dynamic environment. Another problem is to conduct an extensive simu-

Table 1 Comparison of three methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Number of Indexes</th>
<th>Number of Messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>NR</td>
<td>23625.652</td>
<td>3.4532</td>
</tr>
<tr>
<td>RC</td>
<td>11285.268</td>
<td>7.4574</td>
</tr>
<tr>
<td>RC+IN</td>
<td>9011.601</td>
<td>6.7754</td>
</tr>
</tbody>
</table>
lution by considering a more practical situations in which the system consists of heterogeneous peers and the communication time between peers depends on the Round Trip Time (RTT) between them. Moreover, the simulation of this paper shows the total performance of the proposed scheme (including the proposed algorithms described in Sects. 3.2.1 and 3.2.2). We will evaluate the performance of each algorithm and compare the proposed scheme with the view tree system described in Sect. 5. Further, although it seems that the proposed scheme could be applied to other P2P DHTs, we selected CAN which is one of the most popular DHTs as the simulation environment in this paper. We suppose that the performance of the proposed scheme does not change regardless of P2P DHT. We plan to conduct this evaluation on other P2P DHTs and to incorporate the popularity into the proposed mechanism to improve the performance.

Acknowledgment

This work was partially supported by “The Kyushu University Research Superstar Program (SSP)”, based on the budget of Kyushu University allocated under President’s initiative and supported by Grant-in-Aid for Scientific Research.

References


Koji Kobatake received the B.E. degree in electrical engineering, M.E. degree in information engineering from Hiroshima University in 2004 and 2006, respectively. His current research interests include P2P networks and distributed systems.

Shigeaki Tagashira received the B.Eng. degree from Ryukoku University in 1996; and the M.Eng. and D.Eng. degrees in Information Science from Nara Institute of Science and Technology (NAIST) in 1998 and 2000, respectively. He worked as a research associate at Hiroshima University from 2000 to 2007. Since 2007 he has been a research associate professor at Kyushu University. His current research interests include P2P systems, mobile computing and system software. He is a member of the Information Processing Society of Japan (IPSJ) and Institute of Electrical and Electronics Engineers (IEEE).

Satoshi Fujita received the B.E. degree in electrical engineering, M.E. degree in systems engineering, and Dr.E. degree in information science 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.