Peer-to-Peer Netw. Appl. (2012) 5:340–349
DOI 10.1007/s12083-012-0139-5

An efficient search mechanism for supporting partial filename queries in structured peer-to-peer overlay

Guanling Lee · Sheng-Lung Peng · Yi-Chun Chen · Jia-Sin Huang

Received: 30 November 2011 / Accepted: 23 April 2012 / Published online: 15 May 2012

Abstract Accompanying the growth of the Internet, computers throughout the world can connect to each other and exchange information, increasing the convenience and efficiency of information-based work. The advent of data-sharing applications, such as Napster and Gnutella, has made peer-to-peer (P2P) systems popular for widespread exchange of resources and voluminous information between millions of users. In recent years, research issues associated with P2P systems have been discussed widely. To resolve the file-availability problem and improve the workload, a method called the Distributed Hash Table (DHT) has been proposed. However, DHT-based systems in structured architectures cannot support efficient queries, such as a similarity query, range query, and partial-match query, due to the characteristics of the hash function. This study presents a novel scheme that supports filename partial-matches in structured P2P systems. The proposed approach supports complex queries and guarantees result quality. Experimental results demonstrate the effectiveness of the proposed approach.

Keywords Peer-to-Peer overlay · DHT · Partial filename query

G. Lee (✉) · S.-L. Peng · Y.-C. Chen · J.-S. Huang
Department of Computer Science and Information Engineering,
National Dong Hwa University,
Hualien,
Taiwan, Republic of China
e-mail: guanling@mail.ndhu.edu.tw

S.-L. Peng
e-mail: slpeng@mail.ndhu.edu.tw

Y.-C. Chen
e-mail: divien@gmail.com

J.-S. Huang
e-mail: fft16a@gmail.com

1 Introduction

The P2P overlays can be classified as either unstructured or structured. Unstructured P2P overlays, such as Gnutella and Freenet, do not embed a logical and deterministic structure to organize peer nodes. These overlays need a particular message flooding type to search for specific items stored in overlays, resulting in poor efficiency. Several works [1, 2, 6, 8, 10] are proposed to improve these drawbacks by changing search policy or overlay topology. They can ease the network cost effectively; however, file availability is still not solved.

Structured P2P systems, such as CAN [20], Chord [21], Yapper [9] and Tapestry [30], utilize a Distributed Hash Table (DHT) to direct searches to specific node(s) holding the requested data. In DHT-based systems, each node manages a subspace partitioned of the key space, and maintains information about nodes connected as neighbors for use during query forwarding. Files are hashed into values, points in the key space, and published to nodes responsible for the keys. Based on this mechanism, DHT-based P2P systems reduce overhead load and maintain file availability. Moreover, by caching additional neighbor pointers based on peer access frequencies, the average lookup times can be reduced significantly [7]. However, due to the hash characteristic, DHT-based systems can only support keyword searches.

Keyword only searches are insufficient; therefore, supporting general queries in DHT-based systems has been intensively investigated. For example, similarity discovery and skyline query in P2P systems has been investigated in recent years. Similarity discovery is widely employed in conventional document lookup in Internet database systems and search engines. In [11], the distance measurement, called *absolute angle*, is utilized to calculate similarity

between different documents. However, absolute angle do not present similarity between documents well because of the unbounded estimation error. In [22], a method called pSearch was proposed. This method uses the peer vector space model (pVSM) and peer latent semantic indexing (pLSI) to improve the traditional VSM and LSI. However, it would make the dimensions in LSI and CAN mismatch. In [23], a method called rolling-index is used to solve the mismatch problem. However, approaches proposed in [22] and [23] can only be used in CAN. Therefore, Bhattacharya et al. [3] extended the approaches proposed in [22] and [23] and generated a general method that can be used in other structured P2P systems. In [31] a distributed index called the DP-Tree was proposed. Each node in DP-Tree manages a list of relevant documents for popular queries and organizes the document list such that it is searchable within O (logN) time, where N is the total number of participating nodes. In [19], a method called iDistance, which was proposed in [12], was used to reduce the dimensionalities of document vectors to allow similarity queries, range queries, and K-NN queries to be utilized in Chord. Moreover, based on the idea of space partitioning, a novel data structure, LIGhtweight Hash Tree (LIGHT), was proposed in [26] to efficiently support complex query processing in existing DHT-based P2P systems. In the proposed method, a tree summarization strategy was introduced and used to offer each peer a scalable local view. This local view is essentially helpful for distributed query processing.

In [16], a novel framework for processing multi-term queries was introduced. Based on per-query peer-selection strategy using two-dimensional histograms of score distributions, a two-phase peer-selection algorithm was proposed to reduce the communication cost. To improve the efficiency of the method proposed in [16], in [17], for each term, the range of document scores is divided into intervals and a KMV (K Minimal Values) synopsis of its documents is created. By using KMV synopses, the peers can be adaptively ranked according to the relevance of their documents to a given query. In [14], the issue of multiple resource attributes search problem was addressed. By mapping the resources into a multi-dimensional Cartesian space based on the consistent hash values of the resource attributes, FAN method was proposed to support resource queries over multi-dimensional attributes. In [24, 25], the problem of indexing multi-dimensional data in the structured P2P networks was addressed. Based on the idea of kd-tree, a novel indexing scheme called m-LIGHT was introduced. And a clever naming mechanism was employed in m-Light to gracefully map the index tree into the underlying DHT to achieve efficient index maintenance and query processing. Moreover, top-k queries and result ranking problems in a P2P overlay are addressed in [4] and [27], respectively. Two threshold-based top-k algorithms are proposed to optimize the ranking queries in P2P networks. Furthermore, some useful information such as term distributions in local shared contents, user query logs and update frequency are used in the index construction phrase for balancing the trade-off between indexing cost and query processing cost [18, 28]. Moreover, because mobile devises have become indispensable in daily life and a large amount of people use these portable and powerful facilities to share resources and information, the problem of information retrieval in mobile P2P network was discussed in [5]. To tackle the problem, the authors proposed a novel approach by mimicking different human behaviors of social networks to evaluate the distance from a node to certain resources in the network.

A skyline query is a function which returns a set of data objects that is not dominated by any other data objects in the dataset. A data object p_i dominates another data object p_j if $p_i.d_x \geq p_j.d_x$ ($p_i.d_x$ denotes the x-th dimension value of the i-th data object) and is strictly larger in at least one dimension. Skyline queries are particularly useful for the users exploring what is available in the system. In [15], the problem of skyline query in P2P systems was introduced. In the work, the data semantic embedded in semantically structured P2P overlay is exploited to find the cluster whose semantic range covers the optimal attribute value. And, the data that is nearest to the optimal attribute value can be found in the cluster. To alleviate the problem of hot spots presenting in the skyline query processing, in [29], the skyline search space is partitioned adaptively based on query accessing patterns. In order to estimate the query subspaces of the peer nodes, the algorithm control the amount of query forwarding, limiting the number of peers involved and the amount of messages transmitted in the network.

Different from the previous works that discussed about similarity query, range query, multi-attributes query and skyline query, by extending our previous approach [13], this work discusses how to support a general and useful filename *partial match* query in structured P2P systems. Moreover, the comprehensiveness of the proposed approach is also presented in the work. Partial match of a filename search is widely used in Windows and UNIX systems as it is a useful and powerful user function. For example, a query "com*" can retrieve all files whose filename start with "com". "Computer.txt" and "commerce.txt" are examples of retrieved filenames. In the proposed approach, the filename of published files are first translated to form the *index sequences* that can be mapped into a set of keys in a structured P2P system. During query processing, a query is transformed into one or several *query phrase*(*s*) and each query phrase is then mapped into a key in the P2P system structure. By using the key, a user can locate the node responsible for the key. There are some advantages in our work. First, all kinds of file types can be collected. And second, the *recall* of a query can be guaranteed. *Precision* and *recall* are two key statistics about the system's returned

results for a query, and usually used to measure the effectiveness of an information retrieval system. Precision is defined as the proportion of retrieved documents that are relevant and recall is defined as the proportion of relevant documents that are retrieved. In general, there is an inverse relationship between precision and recall. In our approach, all the documents which are relevant to the query will be retrieved.

The remainder of this paper is organized as follows. The problem definition is described in Section 2. Section 3 presents the proposed approach. Experimental results and analysis are discussed in Section 4. Section 5 summarizes this work.

2 Problem definition

Generally, a partial-match query returns data that contain the query keyword. For example, when the keyword "University" is used, all files whose filename contains this keyword are retrieved. To improve query power and efficiency, two symbols, '*' and '+', which mean "nothing or more than one character" and "just one character", are utilized. For example, query "AB*" retrieves all files whose filenames start with "AB." Query "A + B" retrieves all files whose filenames start with "A," end with "B" and have only one character between A and B. With these two symbols, complex queries can be used to attain efficiently. The following strings "AB*F + G" and "*CD + H*Z" are examples of complex queries.

In the proposed approach, the filename of each published file is first partitioned into a set of d-length pieces (d-length indicates that this piece is d long) and each d-length piece is hashed into an *index sequence* $v_0, v_1 \ldots\ldots, v_{d-1}$, denoted as *IS*, with $0 \le v_i \le r-1$ and $0 \le i \le d-1$, where d means *dimension* in the mapping function and r is *range* in each dimension. How to translate a filename into a set of *IS* is discussed in Section 3. In the following, how to map an *IS* into a specific key in Chord is discussed.

Assume m is the size of a finger table, by Eq. 1, an *IS* can be mapped into a specific key in Chord. Similar mapping methods can be utilized to map an *IS* into a specific key in other structured P2P systems.

$$Loc(v_0, v_1 \ldots\ldots, v_{d-1}) = \sum_{j=0}^{d-1} (v_j) r^j \bmod 2^{\mathrm{m}} \tag{1}$$

For example, assume d=2, r=4 and m=4. By Eq. 1, an *IS*<2, 3>is mapped into a specific key: 14. That is,

$$Loc(2,3) = \left(2 * 4^0 + 3 * 4^1\right) \bmod 2^4 = 14$$

Furthermore, if r and d are chosen to satisfy the equation $r^d \bmod 2^m = 0$, load balance can be achieved. The reason is discussed in Section 4.

3 File publishing and query processing

3.1 File publishing

For each published file, the *sliding window partition method* is applied to partition the filename into d-length pieces. Each piece is then put into a *publish function,* as in Eq. 2, and forms an *IS*. In Eq. 2, IS_j denotes the index sequence formed by the d-length piece starting from the j-th character, p is the length of the published filename and h is a hash function such as "SHA-1" [32].

$$\begin{aligned} f(a_i) &= \begin{cases} h[a_i] \bmod \mathrm{r}, \text{if } a_i \neq' +' \\ \text{random value from 0 to r} - 1, \text{if } a_i =' +' \end{cases} \\ IS_j &= f(a_j), f(a_{j+1}), \ldots, f(a_{j+d-1}), 0 \le j \le p-d \end{aligned} \tag{2}$$

According to the above equation, each file can be represented as a collection of its corresponding IS_j, $\{IS_0, IS_1, \ldots, IS_{p-d}\}$ and the collection is denoted as *CIS*. By Eq. 1, each *IS* in *CIS* is mapped into a specific key in Chord. Therefore, each file is mapped into (p-d+1) keys and placed in Chord.

For the case in which the filename length is shorter than d, (d-p) '+' is added to the end of the filename. After appending the filename, the filename length will be d. Hence, only one index is placed in Chord. The reason for assigning a random value from 0 to (d-1) in Eq. 2 is to achieve load balance. That is, when the value is fixed, some peers will have additional workload. Table 1 presents the algorithm in detail.

3.2 Query processing

In the proposed scheme, a section of the query string is selected to represent the query. This selected piece is input into Eq. 3 to form a *query phrase* (*QP*). Given a query S, *QP* is selected as follows. First, S is decomposed into several pieces according to '*'. If the query does not contain any '*', decomposition is unnecessary. By applying the sliding window partition method to all pieces, a set of *QP* candidates is retrieved. If the length of the *QP* candidate is shorter than d, '+' is added based on the position of '*' or at the end when the query does not contain '*', until its length is d. The *QP* candidate that contains the least number of '+' is chosen for input into Eq. 3, and *QP* is obtained.

$$\begin{aligned} m(s_i) &= \begin{cases} h[s_i] \bmod \mathrm{r}, \text{if } s_i \neq' +' \\ -1, \text{if } s_i =' +' \end{cases} \\ QP &=< m(s_0), m(s_1), m(s_2), \ldots, m(s_{d-1}) > \end{aligned} \tag{3}$$

The '+' in the query means "just one character and regardless of which one it is, all characters in that position can be an answer." In Eq. 3, "−1" is used to deal with this situation. When the dimension value is "−1," the whole dimension must

Table 1 File publish

```
Algorithm File Publish
Input :
      p: filename
      d: dimension
Procedure   Publish (p , d)
1: if (p.length ≥ d)
2:   for (int i = 0 ; i ≤ p.length - d ; i++)
3:     s = substring (p , i , i+d -1) //s: viriable to store substring
4:     Translate IS into a key according to equation 1
5:     Publish p in Chord according to the key
6: else   // p.length < d
7:   s = append (d-p.length) '+' at the end of p
8:   Translate IS into a key according to equation 1
9:   publish p in Chord according to the key
10:end
```

be searched. That is, *QP* will be extended into a set of *QP*, denoted as *CQP*, according to the range; for example, when *QP* is <1, 2, 3,-1> and *r* is 4. For the sake of searching the entire dimension, *QP* is extended to {<1,2,3,0>, <1,2,3,1>, <1,2,3,2>, <1,2,3,3>}. Each *QP* in the *CQP* is mapped into a specific key in Chord using Eq. 1. According to the key, the peer responsible for the key in Chord is located.

Notably, because the filenames of published files may be shorter than *d*, if the leading character of the selected QP candidate is '+', the rotation process should be applied to find such a file. For example, when *d*=4, query string "*AB" is transformed into ++AB. For the case in which filename length is less than *d*, such as "AB" or "CAB", cannot be found in the search process. To deal with this

Table 2 Query processing

```
Algorithm  Search
Input :
      Q : query
      d : dimension
Procedure   Query (Q , d)
1:  Select the represented string S in Q
2:  Translate S to QP according to equation 3
3:  if (QP contains "-1")
4:    QP extends to CQP
5:    For each QP in CQP
6:      Translate QP into a key according to equation 1
7:      Put the key into search_pool
8:  else
9:    Translate QP into a key according to equation 1
10:   Put the key into search_pool
11: For each key in the search_pool
12:   Search the  peers responsible for the key
13: end
```

Table 3 Published files and their corresponding keys

Files	Indices (d=4, r=4)	Keys
ABCDEF	ABCD, BCDE, CDEF	(228), (57), (78)
BCDEF	BCDE, CDEF	(57), (78)
CDE	CDE+	(206)
ABCDEG	ABCD, BCDE, CDEG	(228), (57), (142)
ABCFG	ABCF, BCFG	(100), (153)

situation, ++AB is rotated to form the set {++AB, +AB+, AB++} to find all possible files. Table 2 presents the algorithm in detail.

3.3 Examples of query processing

In this section, a detailed example is employed to demonstrate how the proposed method works. In this example, d=4 and r=4.

Table 3 presents the published files and their corresponding keys. Table 4 shows how to map file "ABCDEF" into a set of keys in Chord. The steps in processing query "A + CDE*G*" is presented in Table 5.

3.4 Comprehensiveness

This section shows that the proposed approach ensures that all satisfied files can be found when they exist in a P2P system. That is, the recall of the proposed approach scheme is 100 %.

During query processing, query string S is first decomposed into several pieces according to '*'. Each piece is then partitioned into a d-length string (if its length is longer than d) using the sliding window partition method, and a set of QP candidates is obtained. When the length of the QP candidate is shorter than d, '+' is added to the QP candidate based on the position of '*' or at the end of the QP candidate when the query does not contain '*'. The QP candidate with least number of '+' is selected to represent the query. The following discussions are based on the selected QP (denoted as SQP) candidate.

Table 4 Example of mapping file "ABCDEF" into a set of keys

Steps	Description
1	Published file "ABCDEF"
2	Partition "ABCDEF" into p-d + 1 pieces "ABCD", "BCDE", "CDEF"
3	Put them into translation function $IS_0 = f(A)f(B)f(C)f(D) = 0,1,2,3$ $IS_1 = f(B)f(C)f(D)f(E) = 1,2,3,0$
4	$IS_2 = f(C)f(D)f(E)f(F) = 2,3,0,1$ Map them into a specific key in Chord $0,1,2,3 = \text{key}(228)$ $1,2,3,0 = \text{key}(57)$
5	$2,3,0,1 = \text{key}(78)$ Place index in the peers that are responsible for the specific key

Case 1: '+' is not contained in SQP

The SQP contained in S is transformed into a QP that can be mapped into a definite key. All published files have one identical IS as long as these files contain the SQP. The IS is then translated into the identical key. Therefore, all the published files whose filenames contain the same SQP can be retrieved.

Case 2: '+' is contained in the SQP

'+' means "exactly one character in the position and regardless of which it is." The SQP containing '+' can be concluded in three cases.

I The '+' comes from S. In Eq. 3, '+' is mapped to '−1' and causes QP to be extended into a set of QP for the sake of searching the whole dimension. In other words, all possible values at that dimension are considered in the search process. Therefore, as long as the filename contains the character part of the SQP, the filename can be found.

II The '+' is appended to the QP according to '*'. As '*' means "nothing or more than one character," when the length of QP candidate is shorter than d, '+' is added to the filename based on the position of '*' to represent the situation of "more than one character." As discussed, '+' is mapped to '−1' and causes QP to be extended into a set of QP for searching the whole dimension. By mapping QPs into a set of keys, a search key pool is obtained. When '+' is added at the end of SQP, it will be mapped to certain key contained in the search key pool as long as the filename contains the character parts of SQP. When '+' is added to the front of the character part of SQP, the file whose filename is shorter than d may not be found. As discussed in Section 3.2, the rotation process is applied to deal with this situation. The rotation process enlarges the search key pool containing the search key of the files whose filenames contain the character part of SQP and filename length is shorter than d. Therefore, as long as the filename contains the character part of SQP, it can be retrieved.

III The '+' is added to the end of S when S contains no '*'. When published, the file whose filename is shorter than d, the *blank dimension* is assigned a random number. To identify the file whose filename is shorter than d, "+' is added to the end of S. As mentioned, all possible values in this case are

Table 5 Example of processing query "A + CDE*G*"

Steps	Description
1	Cut query "A + CDE*G*" into pieces according to symbol '*'
	"A + CDE*G*" ➔ "A + CDE," and "G"
2	Process *QP* candidates
	- The length of "A + CDE" exceeds *d* (*d*=4); thus, the sliding window partition method is applied.
	"A + CDE" => {"A + CD", "+CDE"}
	- The length of "G" is shorter than *d*, "+" is added to "G" according to the position of "*" until its length is *d*.
	"G" => {"+++G", "G+++"}
4	The candidate in {"A + CD", "+CDE", "+++G", "G+++"} containing the least number of '+' is selected
	- "A + CD"
6	$QP = f(A)f(+)f(C)f(D) = 0, -1, 2, 3$
7	Extends *QP* to *CQP*
	$CQP = \{0, 0, 2, 3, 0, 1, 2, 3, 0, 2, 2, 3, 0, 3, 2, 3\}$
8	Map each *QP* in *CQP* into a key in Chord
9	$0, 0, 2, 3 = (224)$ $0, 1, 2, 3 = (228)$ $0, 2, 2, 3 = (232)$ $0, 3, 2, 3 = (236)$
	Key (228) can locate a relevant index "ABCD"
	Keys (224), (232) and (236) cannot find any relevant indices.
10	According to index "ABCD," two files can be found.
	ABCDEF
	ABCDEG
11	Only one file, "ABCDEG", matches the user query.
appendix	Precision=1/2 (Two files are obtained and only one file satisfies the query.)
	Recall=1 (One file is relevant to the query in the system and this file is included in the search result.)

considered when searching the whole dimension. Consequently, all files that satisfy the query can be found.

4 Experimental results

4.1 Simulation setup

All programs were written in Java and run on a PC with 3.0G Pentium 4 processor and 1G memory. The published filename is constituted by characters from A–Z and are generated synthetically.

Table 6 Query types

Type	Description
One *	Query contains one star
S	Query wrapped by *
*S	Query start with a star
One +	Query contains one plus
Two +	Query contain two plus

During the simulation, the following metrics are discussed.

1. Load is measured by the number of indices stored in each node. In this simulation, Chord is the system. Therefore, the effect of number of nodes is the same as that in Chord. However, each published file yields $p-d+1$ indices when $p > d$ (p is filename length). Hence, a set of experiments is used to measure the effect of dimension (d) and range (r).
2. Hop-count, measured by the average number of nodes should be accessed when processing a query. In Chord,

Table 7 Default parameter setting

Parameter	Default setting
Number of published files	10×2^{24}
Number of peers participate in the Chord ring	2^{24}
Filename length	Random number from 5 to 25
Query length	Random number from 4 to 12
Finger table size (m)	24
Dimension (d)	12
Range (r)	16

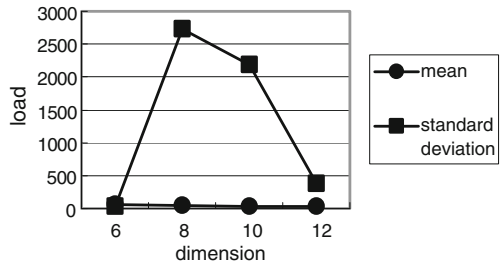

Fig. 1 Effects of dimensions

hop-count is bounded. In the worst case, the average hop-count is m, where m is the finger table size. However, in the proposed technique, when the selected *QP* contains '+', several subqueries are involved to retrieve query results. Therefore, how query types, number of dimensions and range affect hop-counts is discussed.

3. Effectiveness is measured by average precision and recall. Precision and recall are defined in Section 1, and can be calculated as follows.

$$\text{Precision} = \frac{\text{number of relevant files}}{\text{number of relevant indices}}$$

$$\text{Recall} = \frac{\text{number of retrieved files}}{\text{number of total relevant files}}$$

Table 6 shows the query types used in the simulation. And Table 7 shows the default parameters of the simulation.

4.2 Load

Figures 1 and 2 show the effects of dimension and range, respectively. When d is 6 and 12, variance is much smaller than that of others (Fig. 1). This is because when the dimensions are 6 and 12, the relation $r^d \bmod 2^m = 0$ is satisfied. When this condition is not met, some peers must expend additional effort to take responsibility for files whose hash values are larger than $(2^m)^{\left\lfloor \frac{d}{2^m} \right\rfloor}$. Figure 2 presents the same result. The default dimension in Fig. 2 is 12 and the relation is satisfied when range is 4 and 16.

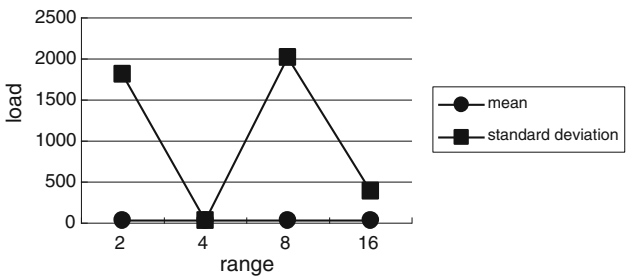

Fig. 2 Effects of range

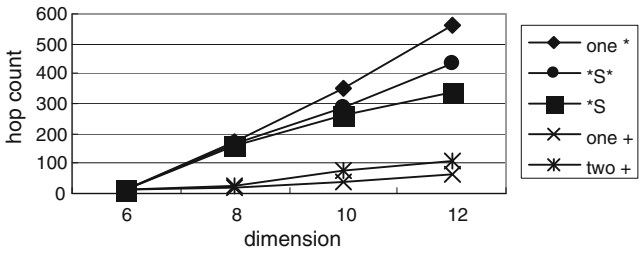

Fig. 3 Average hop-counts with different dimensions

4.3 Hop-count

During the simulation, the *aggregation* method is utilized to route the query. In the proposed algorithm, *QP* is extended to a set of *QP* when *QP* contains −1. To reduce network cost, the aggregation method is used. If the search path of several *QP*s is the same, we only need to traverse the path once. Figure 3 shows the effects of dimension. Regardless of query type, average hop-counts increase as dimension increases. This relationship exists because, during query processing, a query string is first partitioned into a set of pieces according to the position of '*'. As d increases, the piece length has increased likelihood to be shorter than d. As a result, '+' is added to the piece until its length is d. Therefore, *QP* is extended to a set of *QP*, which increases search cost.

Furthermore, the query containing one '*' will incur a large number of hop-counts; the reason is similar to that in the above discussion. That is, according to the query processing method, the query will be partitioned into several pieces according to '*'. For the query containing one '*', it will be partitioned into two small pieces, and as a result, it will have a little chance of being longer than d as d increases. Consequently, '+' will be added to the pieces and *QP* will be extended to a set of *QP*, which increases search cost.

Figure 4 shows the effects of range. Hop-counts increase as range increases. When *QP* contains '+', *QP* is extended to *CQP* according to the range. For example, if the range is 16, the *CQP* will contain 16 *QP*s when the original *QP*

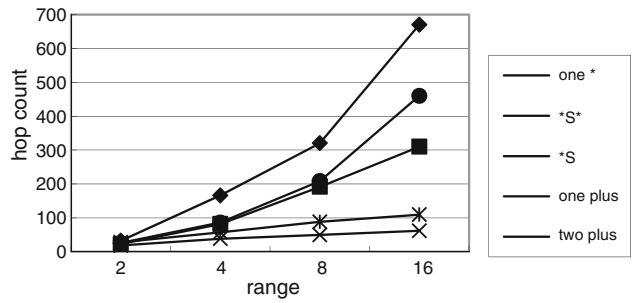

Fig. 4 Average hop-counts with different ranges

contains one '+,' and 16*16 *QP*s when the original *QP* contains two '+.' A large range results in a large search key pool. Therefore, hop-counts increase as range increases.

4.4 Effectiveness

Effectiveness is measured by average precision and recall. As discussed in Section 3.4, the recall of the proposed approach is 100 %. Therefore, only precision is discussed in this section.

Figure 5 shows the precision with different dimensions. The length of *IS* is *d*. A long *IS* improves discrimination in distinguishing between different files. Furthermore, when *d* is increasing, the number of indices yielded by each published file decreases. These two effects cause denominator of precision to decrease. Hence, precision increases as *d* increases.

Figure 6 shows the effect of range. In simulation, the default dimension is 6. Simulation results show that precision increases as range increases. The reason for this relationship is that collision probability of hashing a character into a value decreases as range increases. Consequently, precision increases.

4.5 Summary

According to the simulation results, for all query types, average hop-counts increase as *d*/*r* increases. Moreover, when *r*, *d* and *m* satisfy the equation $r^d \bmod 2^m = 0$, a good load balance can be attained. However, on the contrary, precision decreases as *d*/*r* increases. There is a trade-off between hop-counts and precision. The reason is that a large *d*/*r* improves discrimination in distinguishing between different files but increases the search cost if the length of the query piece is shorter that *d*. We suggest that *d* can be determined according to the query history. That is, to decrease the average hop-counts, *d* should be smaller than the average length of the query pieces to lower the probability that the length of *QP* is shorter than *d*. After that, r can be chosen as a large number to improve the precision.

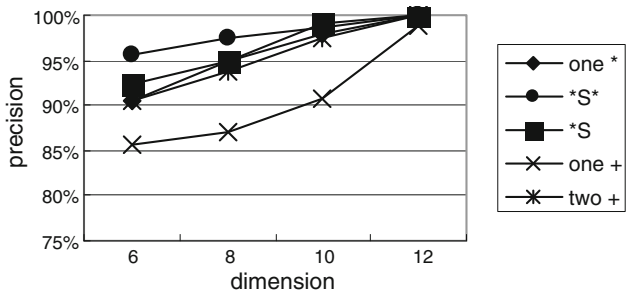

Fig. 5 Precision with different dimensions

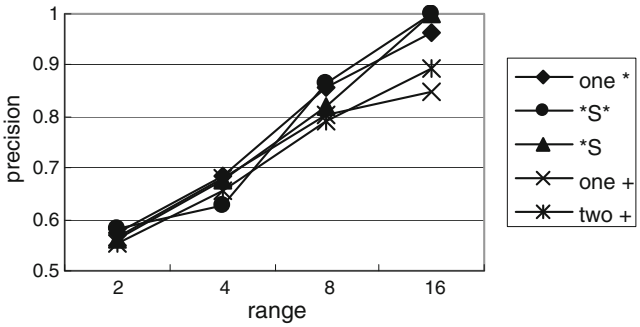

Fig. 6 Precision with different ranges when the dimension is 6

5 Conclusion

This work presented a novel method that supports partial filename queries in a P2P overlay. In the proposed approach, the filenames of published files are first translated to form index sequences that can be mapped into a set of keys in a structured P2P system. During query processing, a query is transformed into one or several query phrase(s), and each query phrase is mapped into a key in the structure P2P system. With this key, a user can find the node responsible for the key. Any structured P2P system employing the proposed approach can support partial filename queries. Additionally, the proposed approach guarantees the recall of queries. Users can find any files they want as long as such files exist. Moreover, a set of simulations is performed to demonstrate the benefit of the proposed approach, and to discuss load balance, network overhead and effectiveness of the proposed method. Simulation results show that when *r*, *d* and *m* satisfy the equation $r^d \bmod 2^m = 0$, an improved load balance is attained. Moreover, increasing *d* and *r* result in high network cost but good precision. Both *d* and *r* should be determined carefully to meet system requirements.

References

1. Bawa M, Manku GS, Raghavan P (2003) SETS: search enhanced by topic-segmentation. In Proceedings of the 26th International ACM Conference on Research and Development in Information Retrieval, Toronto, Canada, pp. 306–313

2. Beverly Y, Garcia-Molina H (2002) Improving search in peer-to-peer networks. In Proceedings of the 22nd International Conference on Distributed Computing Systems, Vienna, Austria, pp. 5–14
3. Bhattacharya I, Kashyap SR, Parthasarathy S (2005) Similarity searching in peer-to-peer databases. In Proceedings of the 25th IEEE International Conference on Distributed Computing Systems, Columbus, Ohio, USA, pp. 329–338
4. Chrysakis I, Chalkidis C, Plexousakis D (2010) Evaluation of top-k queries in peer-to-peer networks using threshold algorithms. In Proceedings of the 19th ACM International Conference on Information and Knowledge Management,Toronto, Ontario, Canada, pp. 1305–1308
5. Chen L, Cui B, Shen HT, Lu W, Zhou X (2009) Efficient information retrieval in mobile peer-to-peer networks. In Proceedings of the 18th ACM Conference on Information and Knowledge Management, Hong Kong, China, pp. 967–976
6. Crespo A, Garcia-Molina H (2002) Routing indices for peer-to-peer systems. In Proceedings of the 22nd International Conference on Distributed Computing Systems, Vienna, Austria, pp. 23–32
7. Deb S, Linga P, Rastogi R, Srinivasan A (2008) Accelerating lookups in P2P systems using peer caching. In Proceedings of the 24th International Conference on Data Engineering, Cancun, Mexico, pp. 1003–1012
8. Doulkeridis C, Nørvåg K, Vazirgiannis M (2008) Peer-to-peer similarity search over widely distributed document collections. In Proceedings of the ACM workshop on Large-Scale distributed systems for information retrieval, Napa Valley California, USA, pp. 35–42
9. Ganesan P, Sun Q, Garcia-Molina H (2003) YAPPERS: a peer-to-peer lookup service over arbitrary topology. In Proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies, Hong Kong, pp. 1250–1260
10. Guclu H, Yuksel M (2007) Scale-free overlay topologies with hard cutoffs for unstructured peer-to-peer networks. In Proceedings of the 27th International Conference on Distributed Computing Systems. Toronto, Canada, pp. 32
11. Hung CH, Chung TK (2003) Similarity discovery in structured P2P overlays. In Proceedings of the 32nd International Conference on Parallel Processing, Kaohsiung, Taiwan, pp. 636–644
12. Jagadish HV, Ooi BC, Tan KL, Yu C, Zhang R (2005) iDistance: an adaptive B^+-tree based indexing method for nearest neighbor search. ACM Trans Database Syst 30(2):364–397
13. Lee G, Huang JS, Chen YC (2010) Supporting filename partial matches in structured peer-to-peer overlay. In Proceedings of the 5th International Conference on Grid and Pervasive Computing, Hualien, Taiwan, pp. 101–108
14. Li R, Song W, Shen H, Xiao W, Lu Z (2011) A flabellate overlay network for multi-attribute search. J Parallel Distr Comput 71 (3):407–423
15. Li H, Tan Q, Lee WC (2006) Efficient progressive processing of skyline queries in peer-to-peer systems. In Proceedings of the 1st International Conference on Scalable Information Systems, Hong Kong, pp. 149–158
16. Mass Y, Sagiv Y, Shmueli-Scheuer M (2009) A scalable and effective full-text search in P2P networks. In Proceedings of the 18th ACM Conference on Information and Knowledge Managemen, Hong Kong, China, pp. 1979–1982
17. Mass Y, Sagiv Y, Shmueli-Scheuer M (2011) KMV-peer: a robust and adaptive peer-selection algorithm, In proceedings of the 4th ACM International Conference on Web Search and Data Mining,, Hong Kong, China, pp. 157–166
18. Nguyen LT, Yee WG, Frieder O (2008) Adaptive distributed indexing for structured peer-to-peer networks. In Proceedings of the 17th ACM Conference on Information and Knowledge Management, Napa Valley, California, USA, pp. 1241–1250
19. Novak D, Zezula P (2006) M-Chord: a scalable distributed similarity search structure. In Proceedings of the 1st International Conference on Scalable Information Systems, Hong Kong, pp. 1–10
20. Ratnasamy S, Francis P, Handley M, Karp R, Shenker S (2001) A scalable content-addressable network. In Proceedings of the ACM SIGCOMM 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, San Diego, USA, pp. 161–172
21. Stoica I, Morris R, Karger D, Kaashoek M, Balakrishnan H (2001) Chord: a scalable peer-to-peer lookup service for internet applications. In Proceedings of the ACM SIGCOMM 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, San Diego, USA, pp. 149–160
22. Tang C, Xu Z, Mahalingam M (2003) pSearch: information retrieval in structured overlays. In ACM SIGCOMM Computer Communication Review, Vol. 33, Issue 1, New Jersey, USA, pp 89–94
23. Tang C, Xu Z, Dwarkadas S (2003) Peer-to-peer information retrieval using self-organizing semantic overlay networks. In Proceedings of the ACM SIGCOMM 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, Karlsruhe, Germany, pp. 175–186
24. Tang Y, Xu J, Zhou S, Lee WC (2009) m-LIGHT: indexing multi-dimensional data over DHTs. In Proceedings of the 29th International Conference on Distributed Computing Systems, Montreal, pp.191-198
25. Tang Y, Xu J, Zhou S, Lee WC (2011) A lightweight multidimensional index for complex queries over DHTs. IEEE Trans Parallel Distr Syst 22(12):2046–2054
26. Tang Y, Zhou S, Xu J (2010) Light: a query-efficient yet low-maintenance indexing scheme over Dhts. IEEE Trans Knowl Data Eng 22(1):59–75
27. Witschel HF (2008) Ranking information resources in peer-to-peer text retrieval: an experimental study. In Proceedings of the ACM Workshop on Large-Scale Distributed Systems for Information Retrieval, Napa Valley, California, USA, pp. 75–82
28. Wu S, Li J, Ooi BC, Tan KL (2008) Just-in-time query retrieval over partially indexed data on structured P2P overlays. In Proceedings of the ACM SIGMOD International Conference on Management of Data, Vancouver, BC, Canada, pp. 279–290
29. Wang S, Ooi BC, Tung AKH, Xu L (2007) Efficient skyline query processing on peer-to-peer networks. In Proceedings of the 23rd IEEE International Conference on Data Engineering, Istanbul, Turkey, pp. 1126–1135
30. Zhao BY, Huang L, Stribling J, Rhea SC, Joseph AD, Kubiatowicz J (2004) Tapestry: a resilient global-scale overlay for service deployment. IEEE J Sel Area Comm 22(1):41–53
31. Zhao DJ, Lee DL, Luo Q (2006) DPTree: a distributed pattern tree index for partial-match queries in peer-to-peer networks. In Proceedings of the 10th International Conference on Extending Database Technology, Munich, Germany, pp. 515–532
32. Available at: http://www.w3.org/PICS/DSig/SHA1_1_0.html

Guanling Lee received the B.S., M.S., and PHD degrees, all in computer science, from National Tsing Hua University, Taiwan, Republic of China, in 1995, 1997, and 2001, respectively. She joined National Dong Hwa University. Taiwan, as an assistant professor in the Department of Computer Science and Information Engineering in August 2001, and became a associate professor in 2005. Her research interests include resource management in the mobile environment, data scheduling on wireless channels, search in the P2P network and data mining.

Sheng-Lung Peng received the B.S. degree in Mathematics from National Tsing Hua University in 1988, and the M.S. and Ph.D. degrees in Computer Science from National Chung Cheng University and National Tsing Hua University in 1992 and 1999, respectively. He is now an associate professor of the Department of Computer Science and Information Engineering in National Dong Hwa University. His research interests include graph theory, algorithms design, data mining, network science and Bioinformatics.

Yi-Chun Chen received the B.S. degree in applied mathematics from Feng Chia University, Taichung, Taiwan, R.O.C., in 2005 and the M.S. degree in computer science from National Dong Hwa University, Hualien, Taiwan, R.O.C., in 2007. He is currently pursuing the Ph.D. degree in the same department. His research interests include data mining and search in the P2P network.

Jia-Sin Huang received the M.S. degree in computer science, from National Dong Hwa University, Taiwan, Republic of China, in 2007. His research interests include data mining and P2P network.