社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2008－ARC－177 (10)
2008－HPC－114 (10)
2008／3／5

Index Distribution Technique for Efficient Search on Unstructured Peer-to-Peer Networks

Sumeth Lerthirunwong†
Naoya Maruyama†
Satoshi Matsuoka†,††
†Tokyo Institute of Technology
††National Institute of Informatics

Resource indexing is an effective technique for fast, successful search on decentralized, unstructured peer-to-peer (P2P) network b s. An index is a summary of resources owned by a node, and is distributed over the P2P network; any node having the index can answer queries on the location of the resources. While more thoroughly distributed indexes can make queries answered more quickly with a small hop count, in large-scale networks, such a scheme may not be always effective due to the large space requirement for keeping indexes at each node. We propose a new index distribution technique that aims to minimize the hop count required for each query by distributing indexes over the network as uniformly as possible, but still in a space-efficient way. To do so, we compute the weight of each index that estimates how many unique resources each index can locate. We give a large weight to an index if it can locate many resources that others cannot. On the other hand, if a resource can be located from an index, we decrease the weights of other indexes that can also locate it. Each node selectively keeps the indexes with the largest weights, thus increasing the chance of successful queries at the node, while keeping the space requirement minimum. Simulation studies show that our distribution technique is effective in decreasing hop counts and messages needed for resolving queries. It decreases the average hop count by up to 44% with 75% less messages when used with flooding based queries. Random-walk with our technique also decreases the average hop count by up to 58% with 82% less messages. Furthermore, the query success rate with a limited timeout condition also increases, approaching nearly to 100%.

I. INTRODUCTION

Recently, most of the popular peer-to-peer (P2P) networks, e.g., FreeNet [16], Gnutella [14], and FastTrack [15], are unstructured since they can scale up very well along with a high demand of users. In such networks, searching resources such as files is one of the most common and important but complicated tasks. It can take long time and generate a large number of messages occupying the overall network; however, it does not always succeed, especially when searching rare resources in large-scale networks. Resource indexing is one of the approaches to the problems [3, 6]. An index is a summary of resources owned by a node, and is distributed over the P2P network; any node having the index can answer queries on the location of the resources on behalf of the resource owner itself. However, while more thoroughly distributed indexes can make queries answered more quickly with a small hop count, in large-scale networks, such a scheme may not be always effective due to the large space requirement for keeping indexes at each node.

We propose a new index distribution technique that aims to minimize the hop counts of queries by distributing indexes over the network as uniformly as possible. We use the Bloom filter [15] to compute the index of a node, which can answer whether a resource is available in the node, but does not always produce correct results. Queries on the existence of a resource succeed with probability; when it fails, we retry the query to find different nodes.

To distribute indexes as uniformly as possible, and at the same time in a space-efficient way, we compute the weight of each index that estimates how many unique resources each index can locate. We give a large weight to an index if it can locate many resources that others cannot. On the other hand, if a resource can be located from an index, we decrease the weights of other indexes that can also locate it. Each node selectively keeps the indexes with the largest weights, thus increasing the chance of successful queries at the node, while keeping the space requirement minimum. This proposed index distribution effectively augments with existing query methods for unstructured P2P networks, such flooding [9] and random walk [7], and decrease their average hop counts.

Simulation studies show that our distribution technique is effective in decreasing hop counts and messages needed for resolving queries. It decreases the average hop count by up to 44% with 75%-less messages when used with flooding based queries. Random walk with our technique also decrease the average hop count by up to 58% with 82%-less messages. Furthermore, the query success rate with a limited timeout condition also increases, approaching nearly to 100%.

II. INDEXING RESOURCES USING BLOOM FILTER

We use Bloom filter [15] to compute an index of a node, which can answer whether a resource is available in the node, but does not always produce correct results. Bloom filter is a space-efficient data structure for a probabilistic representation of a set of objects. Generally, Bloom filter is used to test whether an element is a member of a set. To create a Bloom filter for a set of objects, we hash each object of the set with k hash functions and set the bits corresponding to the hashed results. To check whether an object, x, is a member of the set, we hash x with the same k functions. If all the corresponding bits of the hashed results are set, x may be a member of the set. On the other hand, if there is an unset bit in the corresponding bits, then x is not a member of the set.

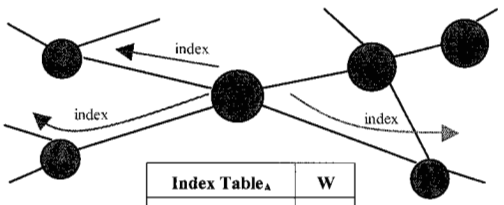
There is a chance that Bloom filter may return a false positive, where all of the corresponding bits are set but the object is not a member. Let m be the length of a Bloom filter, n be the size of the set, and k be the number of hash functions used. Then, the probability of false positive, f, can be calculated as follows [15].

$$f \approx (1 - e^{-kn/m})^k$$

Thus, by setting m and k properly, we can keep the probability of false positive relatively low. For example, suppose the size of the set is 10000, if we set $k = 3$ and m = 1Mbit, then the probability of false positive will be less than 0.002%. In our work, a set of objects corresponds to a set of locatable resources from a node and we use this Bloom filter as the index of the node. Note that the number of set bits correlates with the number of the member objects (resources). Our index distribution technique uses this number to estimate the number of locatable resources.

III. PROPOSED INDEX DISTRIBUTION TECHNIQUE

When a node enters a network, it creates its own index by computing the Bloom filter of its resources and periodically distributes it to neighbor nodes (i.e., active exchange). In addition, when a node answers a query using one of its indexes, it sends back not only the resource location information but also the index itself (i.e., passive exchange). Next time when the requester node queries the same resource, it can resolve it by itself. Receiving indexes from other nodes, it stores them in its index table, and further distributes to neighbor nodes. Figure 1 illustrates a simple example network of nodes. Each circle represents a node and each node has an index table of three entries (including its own index, located in the top-most entry). The more uniformly indexes are distributed over the network, the smaller the average hop count would be. In addition, typical search on unstructured networks uses a Time-To Live (TTL) count as a timeout; thus, making more indexes available within the distance of TTL can increase the search success rate. However, the index table size limits the number of indexes that each peer can hold. Thus, the goal of index distribution is to maximize the number of queries that each peer can resolve within the given table size.



Index Table$_A$	W
01000110	3
11000010	1.5
00101000	2.4

Figure 1. For example network, *node A* hold indexes of other their weight (the first index is its own index). Iteratively, *node A* sends indexes to neighbor node according to our rules

To achieve the goal, we estimate how many new resources an index can locate. We define the estimated number as the *weight* of the index. Each node initially set the weight of its own index by counting the number of set bits in the index. Note that since we compute the indexes by the Bloom filter, the number of set bits correlates the number of resources that the filter includes. For instance, the node A in Figure 1 sets the weight as thee. For indexes distributed from other nodes, we assign the weights using the *new index rule*. We also use two more rules to adjust the weights---the *decay rule* and *integration rule*. The rest of this section describes the rules.

A. New index rule

We use the new index rule to assign initial weights to newly arrived indexes. To do so, the node estimates how many new resources the new index can locate. In other words, if the index has no information on resources that cannot be located only by the existing indexes, we give a minimum weight to the index.

Specifically, we count the number of set bits in the index table with and without the new index. Let S be a function that counts the number of set bits of a given bit sequence, φ_{new} and φ_{old} be the set of indexes in the index table with and without the new index. We compute the weight as follows:

$$W = S(\vee_i \varphi_{new}) - S(\vee_i \varphi_{old})$$

For example, in the sample network shown in Figure 1, *node B* sends its index, 00110001, to *node A*. Then, we compute the weight of the index as follows:

$$W = S(\vee_i \varphi_{new}) - S(\vee_i \varphi_{old}) = 8 - 6 = 2$$

If the index table of *node A* is already full, we drop the index that has the least weight, which is index 11000010 with weight 1.5 in this particular case.

B. Decay rule

Since keeping rarely used indexes in an index table wastes the index space, we eliminate those indexes by decreasing their weights periodically. Specifically, let δ be the *decay rate*, which is a fixed constant between 0 and 1. For each index that has not been used for a certain period, we compute the new weight, W_{new}, as follows:

$$W_{new} = \delta \cdot W_{current}$$

Here, $W_{current}$ denotes the current weight of the index. In our experiment, we set δ to 0.9 and set a period to three index-exchange iterations. For example, in Figure 1, if the third index has not been used for three iterations, we adjust the weight to 2.2.

In some situations, where nodes always query for the similar resources, the decay rate should be set to lower such as 0.5 to increase the average performance of the search because the frequently used indexes will be kept in the index table.

C. Integration rule

This rule optimizes the number of distinctive resources that nearby nodes can locate. Let n and m be a pair of nodes that are directly connected. If both of them have the same index, i, we reduce the weight of the index in either of them. If n resolves a query using, i, m can also resolve it with an additional hop. We trade off a small number of additional hop counts with the diversity of resources locatable using the indexes in nearby nodes.

Specifically, each node distributes a summary of all its indexes to surrounding neighbors by a random walk method with k Time-To-Live (TTL). We summarize indexes by computing the OR of them. Note that since we use the Bloom filter to compute indexes, the value computed by the OR operation still holds the property of the Bloom filter: The summary can return false positive results, but no false negatives are possible. Note also that this summarization produces data of the same size as the original indexes. Thus, in our simulation studies, we consider that the cost of sending a summary is the same as that of a normal index.

When receiving an index summary, the node adjusts the weights of its indexes as follows. For each of the indexes in the table, it applies the AND-operation with the index summary and counts the number of set bits. Let Γ be an index summary and φ_i be an index in the table. We calculate the number as $S(\Gamma \wedge \varphi_i)$. This number estimates how many resources both Γ and φ_i can resolve. We decrease the weight of φ_i, W_i, by subtracting the number, i.e.:

$$W_{i,new} = W_{i,current} - \frac{1}{\varepsilon} S(\Gamma \wedge \varphi_i)$$

Here, ε is the hop count between the node and the summarized node. We divide the estimated number by ε to accommodate the associated cost of additional hop counts.

For example, in the sample network shown in Figure 1, *node C* periodically calculates its index summary and distributes it by using a random walk method with k TTL. Suppose that the index summary of *node C* is 00100111. If *node A*, which is two hops away from *node C*, receives this index summary, it adjusts the weights of all indexes as explained above. Specifically, we update the weight of the second index as follows:

$$W_{2,new} = W_{2,current} - \frac{1}{2} S(\Gamma \wedge \varphi_2) = 2.4 - \frac{1}{2} = 1.9$$

Note that the weight of the second index is reduced because it has the same set bit as the index summary of *node C*.

IV. EVALUATION

A. Simulation Setting

To evaluate the effectiveness of our distribution technique, we evaluate the reduction of average hop counts with indexes distributed by our technique. We implemented an unstructured P2P network simulator and two blind search techniques on top of it, namely flooding and random walk. The simulated network consists of 5,000 nodes with five maximum neighbors

(i.e., five out-degree) for each node. We randomly assigned each node with 4K-bit index, which represents the resources in that node, and used our method to iteratively distribute the indexes over the network. We also implemented another index distribution method that randomly forwards indexes for comparison. We set the index table size of each node to 10.

B. Effects on Increasing the Number of Locatable Resources.

To evaluate the effectiveness in increasing the number of resources locatable with a small hop count, we estimate the increase as follows. At each index-exchange iteration, we choose a set of nodes in the network randomly. For each node, we find the nearby nodes within three hop counts. We compute the OR of the indexes, and use the number of set bits as the estimated number.

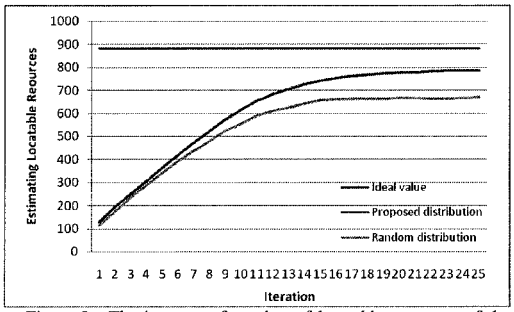


Figure 2. The increase of number of locatable resources of the proposed method compares to random distribution.

Figure 2 compares the estimated numbers between our technique and the random distribution. The ideal value shows the maximum number of set bits within a three-hop-count area. We see that our technique can increase the number of locatable resources significantly compared to the random distribution. After 15 iterations, the numbers nearly stabilized, though they did not reach the ideal value; our technique still exceeded the other by 34.5%.

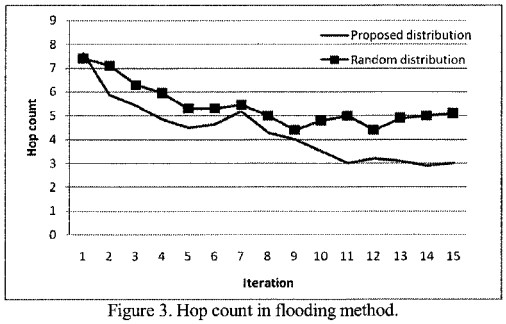


Figure 3. Hop count in flooding method.

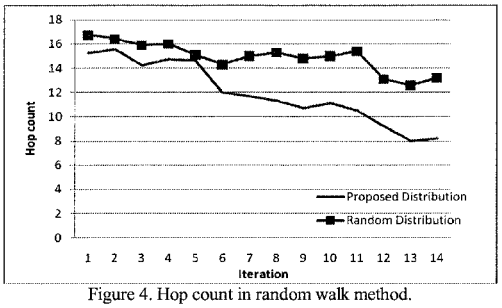


Figure 4. Hop count in random walk method.

C. Effects on Decreasing Hop Counts

To evaluate the effectiveness of our distribution technique in decreasing hop counts of queries, we compare the average hop counts between our technique and the random distribution. We use the flooding and random walk methods to simulate queries.

Figure 3 and Figure 4 show the simulation results. The x-coordinate corresponds to the number of iterations, and the y-coordinate to the average hop count. We see that in both query methods, our proposed technique decreased the average hop counts significantly compared to the initial state. When used with the flooding queries, our technique decreased the average hop count by 44%, which is 40% smaller than that in the random distribution. Similarly, when used with the random-walk queries, it decreased the average hop count by 58%, which is 39% smaller than that in the random distribution.

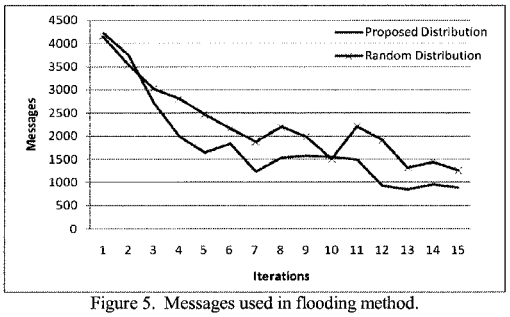


Figure 5. Messages used in flooding method.

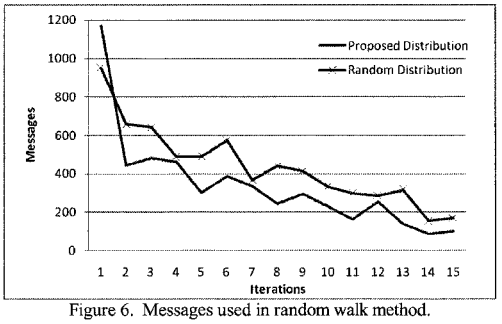


Figure 6. Messages used in random walk method.

Similarly, as in Figure 5 and Figure 6, the message counts used in the search process also decreased significantly with our index distribution, When used with the flooding queries, the number of messages decreased by 75% in 15 iterations, which is 32% smaller than the random distribution. It also decreased when used with the random-walk queries by 82% in 15 iterations, which is 41% smaller than the random distribution.

D. *Effects on Success Rate of Random Walk Methods*

Figure 7 shows that the success rate of random queries also increases. After approximately 12 iterations, the query success rate almost reaches 100% despite some resources is available in less than 5% of all nodes, that because the indexes are distributed effectively around the network.

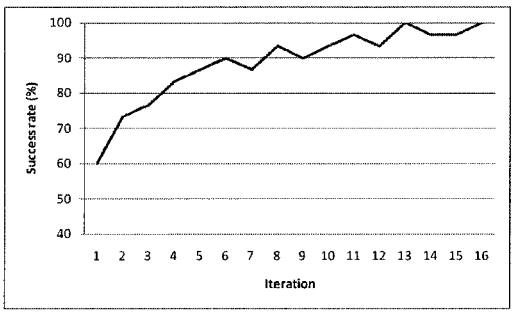


Figure 7. A query success rate of random walk method approaches to 100%.

V. RELATED WORK

A. *Search in Unstructured Peer-to-Peer Networks*

Peer-to-peer networks can be categorized into structured [16, 17,18], and unstructured networks [12,13,14]. While structured networks are generally more efficient in locating resources, they scale poorly because of the need to establish and organize a complete mapping between the index of resources and the locations of the nodes. Moreover, in the centralized network [16], it also has single point of failure problem. On other hand, unstructured peer-to-peer networks are more robust and can scale with the large number of nodes.

Many resource discovery algorithms have been proposed for search in unstructured decentralized P2P networks [10, 11]. They can be categorized into blind search [2, 6, 7] or informed search [4, 8]. Most of blind search algorithms are based on flooding and random walk. Many algorithms have been proposed to improve the efficiency of these flooding and random walk algorithms by adjusting parameters or restricting flooding area. In general, the quality of query results of blind search is still relatively low. For informed search, each node uses hints to facilitate query forwarding. Informed search can be further categorized into proactive and reactive search. Proactive search propagates hints before resources are queried. In contrast, reactive search collects knowledge such as a possible location of a resource from the past query history. In the section three, we mentioned about how indexes are distributed over the network, namely active exchange and passive exchange. The active exchange is comparable to the proactive search because index is a form of hint that is also distributed over the network. On the other hand, the passive condition is comparable to the reactive search. Collecting the indexes that had been requested is like collecting knowledge. Although the reactive search usually have more complicated algorithm such as creating the routing table or applying the learning rule to analyze a behavior of node in the network [4,8], our active exchange is comparable to a simple kind of reactive search. Any of the above-mentioned search techniques can benefit from our index distribution technique in decreasing query hop counts with a limited index table space.

B. *Managing the Overlay Networks in Unstructured Peer-to-Peer Networks*

In order to improve the performance of a search algorithm in unstructured peer-to-peer networks, there are several research projects that proposed methods to build or manage the overlay networks that are appropriate for search algorithm. In [1], the authors proposed a method using learning theory to adaptively manage the overlay network to increase the search efficiency and increase the resilient to targeted attacks on high-degree nodes and maintain search efficiency. The learning theory collects the knowledge from past queries and use the knowledge to manage the over networks. The authors of [6] proposed the probability distribution to increase a success rate of queries for rare items by distributing indexes over the network, which is similar to our proposed distribution. However, our proposed distribution technique also considers the information in each index instead of distributing the indexes randomly. As discussed in section 4, the performance of queries is better when using our proposed index distribution technique compare to random distribution.

VI. LIMITATIONS

A. *Local Optimization Problem*

From Figure 2, our proposed technique cannot increase the number of locatable resources to the ideal value. One of the possible causes is because of a local optimization problem. By distributing indexes according to our proposed method, the estimation of

number of locatable resources may flat into some local maximas. In order to solve this problem and reach the global maxima (the ideal value), some rules in our proposed method need to be adjusted and some addition rules may have to be added into our algorithm. For example, instead of using a fixed k TTL for Random walk to distribute the indexes, we may vary k according to a standard derivation. By occasionally given high value to k, the index will be distributed to the further nodes in the network. Thus, it might solve the local optimization problem.

C. *Bloom Filter Indexes*

In our index distribution technique, we use Bloom filter to create an index because Bloom filter has a property that number of set bits can answer whether a resource is available in the node, although it does not always produce correct results. With this property, we can estimate how many resources the index can locate by counting the set bits of that index. In some situations, the indexes cannot be created using Bloom Filter such as when creating an index for resources with continuous parameters such as CPU speed, memory, etc. In those cases, we have to use another method to estimate how many resources that the index can locate instead of counting set bits.

VII. CONCLUSION

To improve the efficiency of search in decentralized unstructured P2P networks, we proposed a new index distribution technique that attempts to distribute indexes uniformly over networks. We achieve this in a space-efficient way by selectively keep indexes at each node. We prioritize those indexes that can locate many resources that others cannot. Our simulation studies showed that our distribution technique is effective in decreasing hop counts and messages needed for resolving queries. We decreased the average hop count by up to 44% with 75% less messages when used with flooding based queries. Random-walk with our technique also decreases the average hop count by up to 58% with 82% less messages.

In the future, we plan to adjust our proposed distribution technique to further increase the number of locatable resources by solving the above mentioned local optimization problem. We also plan to adapt our distribution technique to enhance the search efficiency in more particulars networks such as networks with power-law distribution.

Acknowledgments This research is supported in part by the MEXT Grant-in-Aid for Scientific Research on Priority Areas 18049028 and the JSPS Global COE program entitled "Computationism as a Foundation for the Sciences."

REFERENCES

[1] Robert A. Ghanea-Hercock, Fang Wang and Yaoru Sun, "Self-Organizing and Adaptive Peer-to-Peer Network", IEEE Transactions on Systems, Man And Cybernetics- Part B, vol.36, no.6, pp. 1230-1236 December 2006

[2] Xiuguo Bao, Binxing Fang, Mingzhen Hu and Binbin Xu, "Heterogeneous Search in Unstructured Peer-to=Peer Networks", IEEE Distributed Systems Online 1541-4922, vol. 6, no.2, February 2005

[3] Jing Wang and Shoubao Yang, "Content-based Clustered P2P Search Model Depending on Set Distance", in Proceeding of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology, pp. 471-476, 2006

[4] Xiuqi Li and Jie Wu, "Improve Searching by Reinforcement Learning in Unstructured P2Ps", Proceedings of the 26th IEEE International Conference Workshops on Distributed Computing Systems, pp. 75, 2006

[5] Haoxiang Zhang, Lin Zhang, Xiuming Shan and Victor O. K. Li, "Probabilistic Search in P2P Networks with High Node Degree Variation", In Proceeding of ICC2007, pp. 1710-1715, June 2007

[6] An-Hsun Cheng and Yuh-Jzer Joung, "Probabilistic File Indexing and Searching in Unstructured Peer-to-Peer Networks", Computer Networks, vol. 50, no. 1, pp. 106-127, January 2006

[7] Christos Gkantsidis, Milena Mihail and Amin Saberi, "Random Walks in Peer-to-Peer Networks", IEEE INFOCOM, pp. 120-130, March 2004

[8] Xiuqu Li and Jie wu, "Cluster-based Intelligent Searching in Unstructured Peer-to-Peer Networks", In Proceedings of the 25th IEEE International Conference on Distributed Computing Systems Workshops, pp. 642-645, June 2005

[9] Qin Lv, Pei Cao, Edith Cohen, Kai Li and Scoot Shenker, "Search and Replication in Unstructured Peer-to-Peer Networks", In Proceedings of the 16th international conference on Supercomputing, pp.84-95, 2002

[10] Dimitrios Tsoumakos and Nick Roussopoulos, "Analysis and Comparison of P2P Search Methods", In Proceeding of ACM International Conference, vol. 152, no, 25, 2006

[11] John Risson and Tim Moors, "Survey of Research towards Robust Peer-to-Peer Networks: Search Methods", Computer Networks: The International Journal of Computer and Telecommunications Networking, vol. 50, no. 17, pp. 3485-3421, December 2006

[12] http://www.gnutella.com/

[13] http://developer.berlios.de/projects/gift-fasttrack/

[14] http://freenetproject.org/

[15] Burton H. Bloom, "Space/Time Trade-offs in Hash Coding with Allowable Errors", Communications of the ACM, vol. 13, no. 7, pp. 422-426, July 1970

[16] http://www.napster.com/

[17] http://www.kazaa.com/

[18] http://www.icq.com/