Adaptive Resource Indexing Technique for Unstructured Peer-to-Peer Networks

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

Searching for particular resources in a large-scale decentralized unstructured network can be very difficult since there is no centralized management to provide the specific location of resources. Moreover, the dynamic behavior of networks and the diversity of user behavior cause the search more complex and may not guarantee success. To address the problems, we propose a new adaptive resource indexing technique that aims to increase both efficiency and quality of the search by reducing both messages and time required for each query. Our approach consists of two complementary techniques. One is an index selection technique that selectively keeps the indices at each peer to increase the chance of successful queries with minimum space requirement. Another is an index distribution technique that automatically adjusts index distribution rate based on the search performance to optimize both the search performance and overhead. We simulate the technique in various network conditions and the results show that our technique is effective in decreasing hop counts and messages needed for resolving queries with only small overhead. It decreases the average hop count by up to 44% with 75%-less messages when used with flooding based queries even facing high churn. 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 networks, e.g., FreeNet [17], Gnutella [16], and FastTrack [15], are unstructured and decentralized networks since they only enforce minimal constraints on the network topology, which critical in a highly dynamic environment, and can scale up 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. Since there is no centralized management to provide the location of each resource, the search is typically done by flooding queries thoroughly the network. Thus, it can take a long time and generate a large number of messages occupying the overall network links. Moreover, it cannot guarantee a success of search, especially when searching rare resources in large-scale networks. One common approach to the search on unstructured peer-to-peer networks is Resource Indexing [4]. An index of a peer is a summary of resources owned by the peer, and is distributed over the peer-to-peer network; any peer having the index can answer queries on the location of the resources. While more thoroughly distributed indices can make queries answered more quickly with a small hop count, in large-scale networks, such a scheme may not always be feasible due to the large space requirement for keeping indices at each peer; it may be impossible for each peer to keep all indices in the entire network. Furthermore, distributing indices constantly without considering the stationariness of networks can waste a lot of messages since not all resources are dynamic.

Our goal is to increase overall search efficiency and success rates in the decentralized unstructured peer-to-peer network. To that end, we propose a new adaptive resource indexing technique that aims to reduce space and network overheads. We have presented a preliminary version of our index selection technique [5], where each peer selectively keeps the indices to maximize the search efficiency. This paper extends the work by an improved index selection technique and a new adaptive distribution technique. In the index selection technique, we compute the weight of each index that estimates how many unique resources each index can locate. We also present several rules that adjust index weight in accordance with changing behavior of networks and users. Each peer selectively keeps the indices with the largest weights, thus increasing the chance of queries to be solved by the peer. In the adaptive index distribution, each peer automatically adjusts its index distribution rate based on the performance of the search to optimize both overhead and search performance.

We simulate the proposed technique in various network conditions such as high-churn networks and scale-free networks with variable popularity of resources. The simulation results show that our technique is effective in decreasing hop counts and messages needed for resolving queries with small overhead. Furthermore, the query success ratio with a limited timeout condition also increases significantly.

II. RELATED WORK

A peer-to-peer network can be categorized into either structured networks or unstructured networks. A structured network has a strict control on an underlying network structure, a content publication strategy and query routing. While it is suitable for distributed storage systems involving contents with

unique identifiers, it has a high management cost for maintaining the topology of the network when peers enter or leave the network and it is also sensitive to failures, errors and malicious peer behaviors that are frequent in recent large-scale peer-to-peer environments. The examples of these networks are Chord [7], CAN [8], and Tapesty [9]. All these networks utilize the distributed hash table (DHT) for fast and accurate resource lookup but have different network topologies.

On the other hand, the unstructured network has minimal constraints on the network topology and content distribution, so it is more suitable in highly dynamic and often ad-hoc peer-to-peer environments. Therefore, most of popular peer-to-peer applications nowadays operate on unstructured networks, e.g., Gnutella [15] and Freenet [17]. In these networks, peers connect to other peers in an ad-hoc fashion. The location of the resources is not controlled by a single centralized entity in the system. There are no guarantees of query solution. Typically these networks use blind search or informed search, according to whether peers utilize some information to locate desired resources or not.

In case of the blind search method, peers in the network have no information or hint regarding resource locations. Thus, peers have to propagate the query to a sufficient number of peers in order to be able to find a desired resource or to resolve the query. Blind search approaches merit from its simplicity. It also has relatively small cost dealing with peer leaving or joining the network so it suits for the network with high churn. On the other hand, by propagating the queries over the network, a lot of messages are generated and occupying overall network, which can lead to scalability problems. Many blind search techniques have been proposed, such as Random Walk [3, 9] and Iterative Deepening [4]. Unlike the blind search method, the informed search method utilizes information about resource locations that help route queries to the desired resources and thus reducing unnecessary messages. The semantics of the used information can be very simple, for example, forwarding hints to exact resource locations. Examples of these techniques are Intelligent-BFS [13] and Local Indices [14]. Resource Indexing can also be categorized into a kind of the inform search since the index can be considered a hint of the resource. But it can also be effectively augmented with existing blind methods such as flooding and random walk, and decrease their average hop counts for query according to our simulation studies. There are also several research projects that proposed methods to build or manage the overlay networks that are appropriate for search algorithm. In [11], 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.

Aside from search methods and overlay management algorithms, there are several studies which relates to our work. In [1], the authors show that a random replica distribution strategy can achieve the optimal results through a mathematical proof. Although this method can estimate a reasonable number of replicas in the system to achieve nearly perfect query success rate, theory to distribute them is not explained and the cost of distribution is not considered. Probabilistic file indexing [2] applies the resource indexing scheme as in our proposal. The authors proposed the probability distribution to increase a success rate of queries for rare items by distributing indices uniformly over the network. However, our technique also considers the information in each index instead of distributing the indices randomly and achieves better search results according to our simulation studies.

III. RESOURCE INDEXING SCHEME

In the resource indexing scheme, when a new peer joins the peer-to-peer network, it has to create an index which describes or tells the locations of its resources. Basically, the index is a summary or a metadata of resources which are owned by a peer. This index will be distributed over the peer-to-peer network; any peer having this index can answer queries on the location of the resources on behalf of the resource owner.

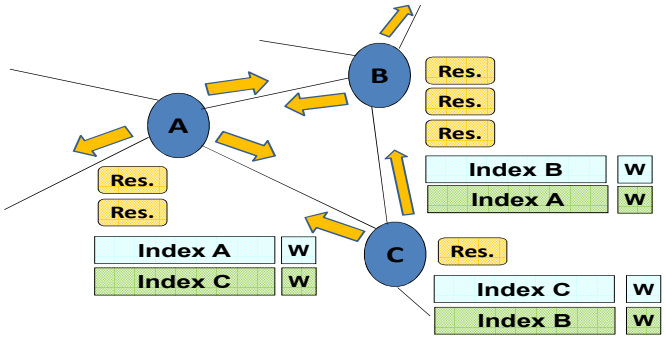

Figure 1. Each peer holds resources and indices. Iteratively, each peer sends indices to neighbor peer according to our technique.

Figure 1 illustrates a model network for resource indexing; each peer has its own resources, its own index and also other peers' indices. Iteratively, each peer distributes its index to the neighbor peers. In the following sections, we describe how to create and utilize the index, and then we explain in details our index selection and distribution techniques.

Index Creation and Utilization

The index can be created by various methods depending on resources to index. Each method suits for specific purpose and application. For example, in a file-sharing system, *peer A* creates its index from all of its files, such as {ubuntu.iso, eclipse.zip …}. If *peer B* has an index of *peer A*, *peer B* can resolve query 'eclipse.zip' on behalf of *peer A*. In case of our research, the index should be able to answer the existence of specific resources, and thus we choose to use a Bloom filter [6] to create our index.

Bloom Filter

Technically, *Bloom filter* is a space-efficient data structure for a probabilistic representation of a set of objects. Bloom filter is used to test whether an element is a member of a set.

The index created using Bloom Filter can answer whether a resource is available in the peer, but does not always produce correct results. 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 cannot be a member of the set. Figure 2 depicts an example. *Peer A* has two resources. Both resources are hashed and result in bit array which can represent existence of both resources.

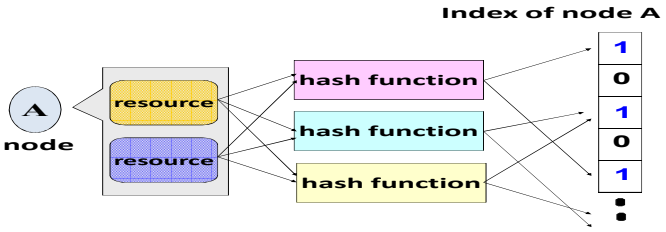

Figure 2. The example of using Bloom Filter, *peer A* creates Bloom Filter which describes its two resources.

When each peer distributes these Bloom Filters into the network, it has to enclose the Bloom Filter with other information such as an IP address and port of the owner and the TTL or the time-to-live of the index which is left.

Index Usage

When each peer successfully creates its index and distributes it into the network, this index will be used by other peers to resolve their query. When searching for a desired resource, query is created by a desired filename or keyword. This query is distributed over the network according to search algorithm (such as flooding or random walk). Once a peer receives this query, it searches its own resources and then searches through all indices in its index table using the same hash function as index creation. Depending on whether the query can be resolved or not, each peer proceeds as follow;

- Irresolvable case - If there is no match found, peer will continue forwarding query according to search algorithm.
- Resolvable case - If the resource is found or the location of resource is found, peer will return resource information and index to the requester. According to search algorithm, some algorithm may want peer to continue forwarding query even when the desired resource is found.

Because the index table of each peer has a limited space, we use our proposed index selection technique to selectively keep indices in the index table to maximize the search efficiency while keeping space-requirement minimum (Section IV). Furthermore, in order to keep the overhead of resource indexing minimum, we apply our adaptive index distribution technique (Section V). The technique automatically adjusts the index distribution rate to optimize the overhead. In the next section, we discuss in details about these two techniques

IV. PROPOSED INDEX SELECTION TECHNIQUE

For the resource indexing scheme, the more uniformly indices are distributed over the network, the smaller the average hop count would be for the queries. In addition, typical search on unstructured networks use a Time-To Live (TTL) count as a timeout; thus, making more indices available within the distance of TTL can increase the search success rate. However, the problem is that the index table size of each peer limits the number of indices that each peer can hold. Thus, the goal of index selection technique is to choose which indices to keep that will maximize the number of queries that each peer can resolve within the given limited table size.

To achieve the goal, we estimate how many new resources an index can locate. We use the estimated number as the *weight* of the index. Each peer initially set the weight of its own index by counting the number of set bits in the index. Since we compute the indices by the Bloom filter, the number of set bits correlates with the number of resources that the filter includes. For indices distributed from other peers, we assign the weights using the *new index rule*. We also use two more rules to adjust the weights; the *usage rule* and *integration rule*, which serve for different purposes which make the technique more adaptive. The rest of this section describes the rules.

A. New index rule

We use this rule to assign initial weights to newly arrived indices. To do so, the peer 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 indices, 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 indices in the index table with and without the new index. We compute the weight as follows:

$$W = S(\vee_i \varphi_{new}) - S(\vee_i \varphi_{old}) \quad (1)$$

For example, in the sample network shown in Figure 1, *peer B* sends its index, 00110001, to *peer A*.

Table 1. The index table of *peer A*.

Index Table	Weight
01000110	3
11000010	1.5
00101000	2.4

Let the table 1 depicts the index table of *peer 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 *peer A* is already full, we drop the index that has the least weight, which is the index with weight 1.5 in

this particular case.

B. Usage rule

When considers the real world network, the popularity of each resource is different; some resources may be interested or requested by a lot of users while some resource may be requested by only a handful of users. Moreover, as timed passed, popular resources can become rarely used resources and rarely used resources can become popular resources. The objective of our usage rule is to make indices of these popular resources easier to find so that the query for these resources can be resolved faster. To do so, we give more weight to the index when it has been used, while the weight of the index which has not been used will be reduced periodically since keeping rarely used indices in an index table wastes the index space. Specifically, let δ be the *decay rate*, which is a constant between 0 and 1. Let θ be the *usage constant*, which is also a constant. For each index that has been used or not been used for a certain period, we compute the new weight, W_{new} , as follows:

$$W_{new} = \begin{cases} \delta \cdot W_{current} & ; \text{unused} \\ W_{current} + \theta & ; \text{used} \end{cases} \quad (2)$$

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 the Table 1, if the third index has not been used for three iterations, the weight is seduced to 2.2. We discuss the effect of this rule later in the evaluation part.

C. Integration rule

This rule optimizes the number of distinctive resources that nearby peers can locate. Let *n* and *m* be a pair of peers 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 indices in nearby peers. Specifically, each peer distributes a summary of all its indices to surrounding neighbors by a random walk method with *k* Time-To-Live (TTL). We summarize indices by computing the bitwise logical OR of them. Note that since we use the Bloom filter to compute indices, 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 indices. 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 peer adjusts the weights of its indices as follows. For each of the indices 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.:

$$W_{i,new} = W_{i,current} - \frac{1}{\varepsilon} S(\Gamma \wedge \varphi_i) \quad (3)$$

Here, ε is the hop count between the peer and the summarized peer. We divide the estimated number by ε to accommodate the associated cost of additional hop counts. For example, in the sample network illustrated in Figure 1, if there exist *peer D,* which located two hops away from *peer A*. *Peer D* periodically calculates its index summary and distributes it by using a random walk method with *k* TTL. Suppose that the index summary of *peer D* is 00100111. If *peer A*, receives this index summary, it adjusts the weights of all indices 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 *peer D*.

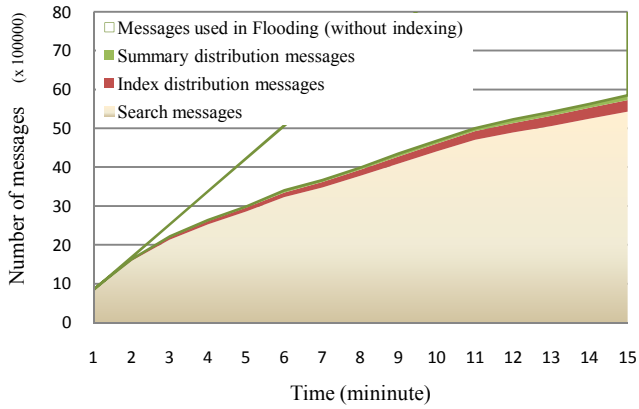

Figure 3. The cumulative messages generated in the search process which consists of messages used in the index distribution and the query messages.

V. PROPOSED INDEX DISTRIBUTION TECHNIQUE

Although both time and number of messages used to resolve search queries can be greatly reduced by utilizing the resource indexing scheme, there is an overhead which causes by this technique. Figure 3 illustrates the cumulative messages which are generated throughout a search process within 15 minutes period. The graph shows that number of messages used in Flooding method with and without resource indexing scheme. The green and red area is the overhead causes by the resource indexing technique. Despite the overheads account only about

5% of the messages used to resolve the search queries, these overheads continuously increase in a linear way. This is because of the indices are distribution constantly throughout the whole operation time. Moreover, the messages used in search are going to decrease, so the proportion of the overhead will also increase. Therefore, the objective of our adaptive index distribution technique is to optimize both number messages used in search and the overhead cause by the resource indexing while still maintains the search efficiency.

A. Self-Adjusting index distribution

According to our simulation results, if each peer iteratively distributes indices over the network using the resource indexing technique, the number of messages used in search algorithm will be reduced. Approximately, after ten iterations of distribution, the average number of messages used for resolving each query will become stable. Our technique utilizes this fact and forces each peer to monitor the search performance or simply an average number of the messages used for resolving each query. When a peer detected that the number became stable, the peer will temporary stop the index distribution. However, by the dynamic nature of peer-to-peer network, some peers may leave or join the network. Resources at each peer are also changed. For this reason, if indices are not updated or distributed for some period of time, the efficiency of resource indexing will be decreased and the messages used for resolving queries will be increased. Thus, if peers detected that the number of query messages has increases, the peer will restart the index distribution again as illustrated in Figure 4.

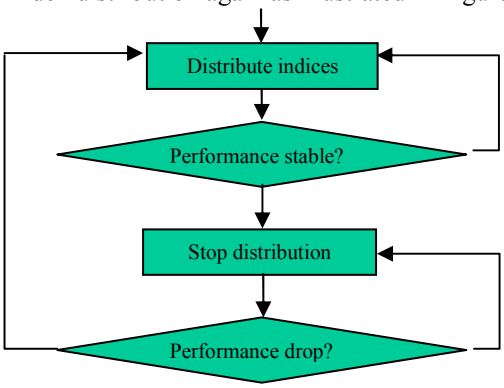

Figure 4. The process of adaptive index distribution technique.

B. Performance monitoring & start-stop condition

To be able to adaptively adjust the distribution rate, each peer has to monitor the search performance and then use the result to adjust the index distribution rate. In our work, we use the number of query messages as an indicator for search performance. If the average number of query messages used for resolve any query is reduced to stable, the search performance is assumed to be stable.

The monitoring process has to collect the average number of messages used in i^{th} period. Each peer has to collect this number from the peers within k hop count. In the evaluation section, we investigate the proper value of k which can use to estimate the performance of the entire network. Let n_i be an average number from peers within k hop count. The distribution of indices will be stopped if the following condition is satisfied.

$$\left(\left(\frac{n_i + n_{i-1}}{2}\right) - \left(\frac{n_{i-1} + n_{i-2}}{2}\right) < \delta\right) \;\&\&\; (n_i < n_{i-1} < n_{i-2}) \quad (4)$$

Where δ is a threshold constant, which in our simulation is set to 50. The value can be changed depending on the size of network, the size of index table and the search algorithm. After the index distribution has stopped, a peer still continuously monitors the number of query messages. When the following condition is satisfied, the peer starts the distribution again.

$$n_i > n_{i-1} > n_{i-2} > n_{i-3} \quad (5)$$

Then, the peer repeatedly checks these two conditions throughout the whole operation. Figure 4 illustrates a diagram chart which shows the flow of each peer.

VI. EVALUATION

To evaluate the effectiveness of our technique, we evaluate the reduction of average hop counts and number of messages for indices distributed by our technique. We implemented an unstructured peer-to-peer network simulator and two blind search techniques on top of it, namely flooding and random walk. In the first part, we use a random network to investigate efficiency and overhead of the index selection rules as well as the adaptive index distribution. Next, we simulate various network situations to investigate more about the effect and the robustness of our technique. We simulate the situation where there exist the popular and rarely used resources like real-world networks, a high churn network and also a scale-free network.

Table 2. Simulation settings.

Network size	5,000 peer
Network type	- random network (5 maximum neighbor) - scale-free network
Churn rate	- 30 peer/min - 100 peer/min
Query rate	0.3 query/min
Query type	- random query - weighted query
Index type	4K Bloom filter index (5 maximum set-bit)
Index table size	10
Search Method	- Flooding - Random Walk (k=3)

A. Effects on the random network

In this section, we investigate the effect of our technique on the random network. The simulated random network consists

of 5,000 peers with five maximum neighbors (i.e., five out-degrees) for each peer and churn rate is set to 30 peer/minute. We assign each peer with 4K-bit index, which represents the resources in that peer. The index will be distributed according to the resource indexing scheme. We set the index table size of each peer to 10. The frequency of query for each peer is set to 0.3 query/minute. We implemented another index distribution method that randomly forwards indices for comparison with our technique. The simulation settings are summarized in Table 2. First, we evaluate the effectiveness in increasing the number of resources locatable within a small hop count. Since we assign the index to each peer, not the real resources, the number of locatable resources is estimated from the index. At each index-exchange iteration, we choose a set of peers in the network randomly. For each peer, we find the nearby peers within three hop counts. We compute the OR of the indices, and use the number of set bits as the estimated number.

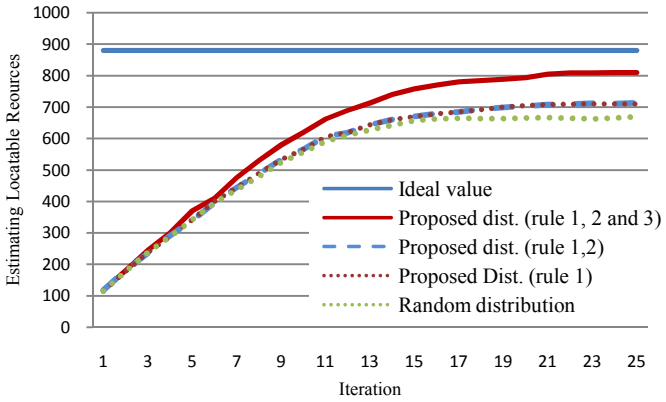

Figure 5. The number of locatable resources-increasing rate of the proposed method compares to random distribution.

Figure 5 compares the estimated numbers of locatable resource 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. When applied only the first rule, the number of locatable resources increases about 9.3%. Consequently, we applied the *usage rule*, however, the result did not show any improvement due to the objective of the *usage rule* is not to increase the number of locatable resources. The real effect of the usage rule is investigated in the next section where we simulate the popular and rarely used resources. Finally, we applied all rules to the system, though the result did not reach the ideal value; the number of locatable resource increase significantly by exceeding other by 34.5%.

Next, we evaluate the effectiveness of our 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 6.The average hop count in Flooding method using our proposed technique compare with random distribution technique.

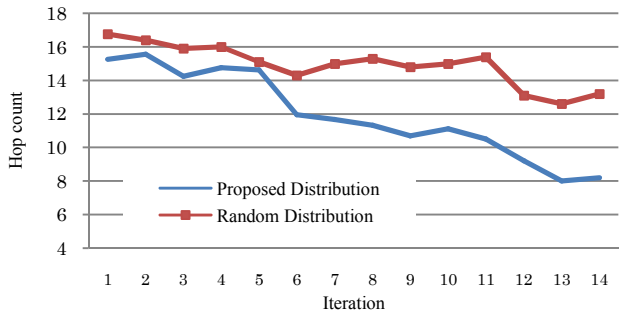

Figure 7.The average hop count in random walk method using our proposed technique compare with random distribution technique.

Figure 6 and figure 7 show the results from the simulation. 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 can decrease 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.

Similarly, 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 33% 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.

B. Effects of adaptive index distribution technique

We evaluate the effectiveness of our adaptive index distribution technique by considering the reduction of the overhead and the preservation of the search performance. First, we investigate the value of k which can be used for estimating the search performance of the entire network efficiently. We compare the difference of the average number of messages used to resolve query within k hop count. Note that zero hop count means that peer using only local information within the peer. Figure 8 shows that the value from only two hop counts can be used to efficiently estimate the search performance of the entire network. The difference of between the value from

two hop count and the value from the real entire network is different by less than 7%.

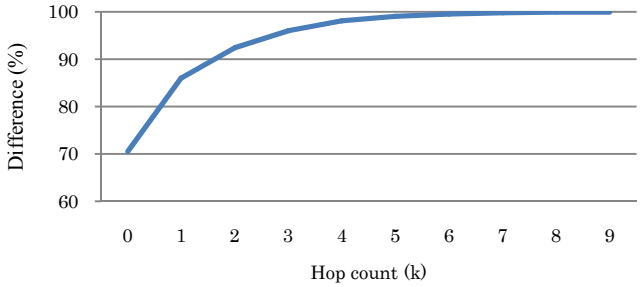

Figure 8. The difference between average value from the peer within *k* hop count and the real value from the entire network

Next, we evaluate the efficiency of our distribution technique. Figure 9 shows the overhead caused by resource indexing which is the number of messages used for distributing the indices. The result shows that our technique can reduce the overhead significantly by more than 40% and the percentage tends to increase since the number of messages used in the normal distribution grows linearly.

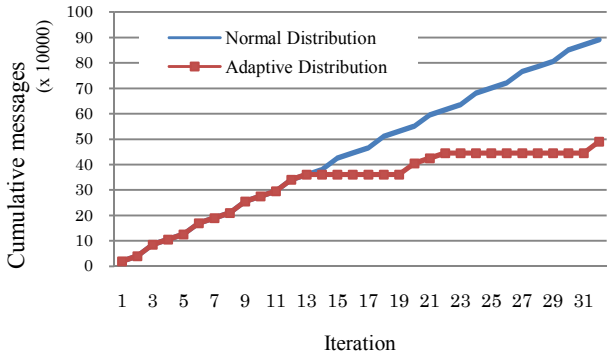

Figure 9. Hop count in random walk method.

Figure 10 shows the number of average hop count when applying our adaptive distribution. The result shows that the search efficiency is almost as the same as the normal distribution, while the overhead has been reduce greatly as previously shown. The hop count only increases a little bit in the green circle area which is the period when the system temporary stops the distribution of indices.

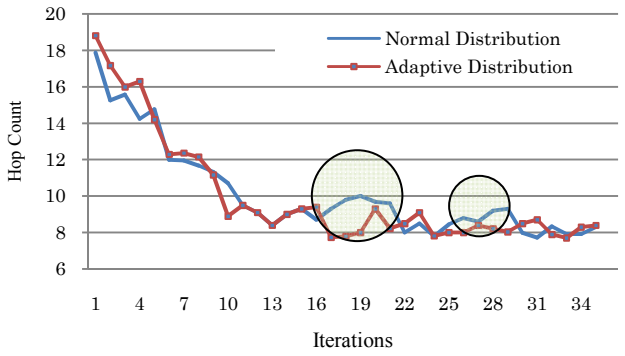

Figure 10. Hop count in random walk method.

C. *Effects on popular resources and rarely used resources*

In this section, we evaluate the effect of the *usage rule*. We simulate the situation where there exist popular resources and rarely used resources. We divide all resources into three categories; *popular resources*, *normal resources* and *rarely used resources*. In case of the popular resources, 70% of all queries will request these resources. In case of the normal resource, 25% of all queries will request these resources. And the other 5% of all queries will request the rarely used resources. We use the same simulation settings as in the previous section except for the query. Figure 11 shows the simulation result of the search using random walk method. The result shows that by applying our technique the hop count used for search can be greatly reduced. We compare the result when applied and did not apply the *usage rule*. By applying this rule, the peers gradually learn which resources are the popular ones, and the search efficiency can be further increased greatly. According to the result, the hop count can be further reduced by more than 24.4%.

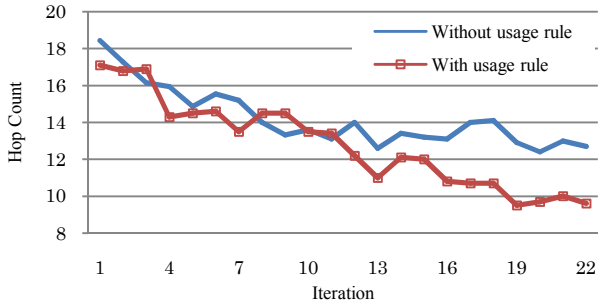

Figure 11. Hop count in random walk method with weighted query.

D. *Effects on high-churn networks and scale-free networks*

Finally, we evaluate the robustness and efficiency of the proposed technique in the real-world-like network characteristic. In a large-scale network, it is common that the churn rate of the network is high. The high churn usually causes problems to both search algorithms and topology management algorithms. Although churn seems to have less impact on unstructured network than structured network, the search efficiency is reduced greatly especially for the informed search. We set the simulation to evaluate the effect of high churn on the search efficiency. We set the churn rate in our experiments twice as high as that observed in Gnutella [15]. Figure 12 shows the number of locatable resources within three hop count. According to the result, when facing high churn, the number of locatable resource using our technique drop about 13%. However, the result is still better than the result of the random distribution by more than 15% and thus confirms the robustness of our technique to the high churn network

Scale-Free Networks are complex networks, where their structure and dynamics are independent of the network size. Its degree distribution follows a mathematical function called a power law, i.e.,

$$P(k) \sim k^{-\gamma}$$

Where *P(k)* is the probability that a peer in a network connects with *k* other peers. The power law state that *P(k)* is roughly proportional to $k^{-\gamma}$, where γ varies approximately from two to three in many real networks. In case of Gnutella, there is a study which confirmed that the presence of a power-law distribution and γ is close to 2.3 [14]. We evaluate the efficiency and robustness of our technique in the scale-free network by setting the simulation as the previous section except for the network type, which is the scale-free network with γ= 2.3. Figure 12 shows the number of locatable resources within three hop count. According to the result, when network is scale-free network, the number of locatable resource using our technique is increased by 4% compared to the random network case, and by 14% compared to the random distribution. This also confirms the robustness of our technique in the scale-free network. The property of the scale-free network (average radius of networks is shorter than random networks when network is the same size) may lead to the better achieved results.

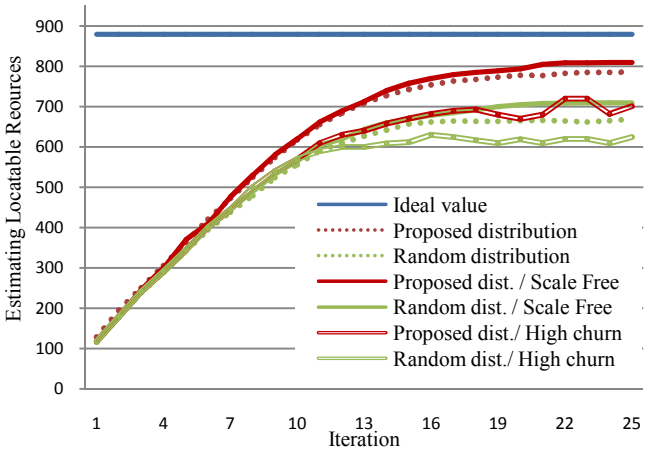

Figure 12. The number of locatable resources-increasing rate of the proposed method compares to random distribution on the high-churn network and scale-free network.

VII. CONCLUSION

To improve the efficiency of search in decentralized unstructured peer-to-peer networks, we proposed a new resource indexing technique that attempts to distribute indices uniformly over networks with minimal overhead. We achieve this in a space-efficient way by selectively keeping indices at each peer. We prioritize those indices that can locate many resources that others cannot and also automatically adjust distribution rate to minimize the overhead. Our simulation studies showed that proposed indexing 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. Moreover, the result from various network conditions also confirmed the efficiency and the robustness of our technique.

In the future work, we plan to adapt our technique to be suitable in the hybrid peer-to-peer network [1]. The parameters of our technique, such as decay rates and usage constants, should be automatically adjusted in accordance with user behavior. We also plan to evaluate the proposed technique with a wider variety of search algorithms and network conditions.

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] Xucheng Luo, Zhiguang Qin, Jinsong Han and Hanhua Chen, "DHT-assisted Probabilistic Exhaustive Search in Unstructured P2P Networks", in Proceeding of the IEEE International Parallel & Distributed Processing Symposium (IPDPS), 2008

[2] 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

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

[4] 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

[5] Sumeth Lerthirunwong, Naoya Maruyama and Satoshi Matsuoka, "Index Distribution Technique for Efficient Search on Unstructured Peer-to-Peer Network", In Proceeding of the 2008 ECTI International Conference, pp. 97-100, May 2008

[6] 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

[7] Ion Stoica, Robert Morris, David Karger, Frans Ka~shoek, and Hari Balakrishna, "Chord: A scalable peer-to-peer lookup service for internet applications", In Proceedings of SIGCOMM, pp. 149-160, 2001

[8] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp and Scott Shenker, "A Scalable Content-Addressable Network", Proceedings of ACM SIGCOMM, pp. 161-170, 2001.

[9] Ben Y. Zhao, John Kubiatowicz, and Anthony D. Joseph, "Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing", Report No. UCB/CSD-01-1141, April 2001

[10] 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

[11] 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

[12] V. Kalogeraki, D. Gunopulos, and D. Zeinalipour-Yazti, "A Local Search Mechanism for Peer-to-Peer Networks," in Proceeding of 11th International Conference on Information and Knowledge Management (CIKM), 2002.

[13] B. Yang and H. Garcia-Molina, "Improving Search in Peer-to-Peer Networks," in Proceeding of the 22nd International Conference on Distributed Computing Systems, 2002

[14] Saroiu S., Gummadi P. K., and Gribble S. D, "A Measurement Study of Peer-to-Peer File Sharing Systems", In Proceedings of Multimedia Computing and Networking (MMCN'02), San Jose, CA, January 2002

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

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

[17] http://freenetproject.org/