An Algorithm to Reduce the Communication Traffic for Multi-Word Searches in a Distributed Hash Table

Yuichi Sei[1], Kazutaka Matsuzaki[2], and Shinichi Honiden[3]

[1] The University of Tokyo Information Science and Technology Computer Science Department, Tokyo, Japan sei@nii.ac.jp
[2] The University of Tokyo Information Science and Technology Computer Science Department, Tokyo, Japan matsuzaki@nii.ac.jp
[3] National Institute of Informatics, Tokyo, Japan honiden@nii.ac.jp

Abstract. In distributed hash tables, much communication traffic comes from multi-word searches. The aim of this work is to reduce the amount of traffic by using a bloom filter, which is a space-efficient probabilistic data structure used to test whether or not an element is a member of a set. However, bloom filters have a limited role if several sets have different numbers of elements. In the proposed method, extra data storage is generated when contents' keys are registered in a distributed hash table system. Accordingly, we propose a "divided bloom filter" to solve the problem of a normal bloom filter. Using the divided bloom filter, we aim to reduce both the amount of communication traffic and the amount of data storage.

1 Introduction

Peer-to-peer systems are distributed networks that can share contents or services without the need for a central server. The first peer-to-peer systems, such as Napster [5] and Gnutella [1], lacked scalability. Distributed hash table (DHT) systems such as Chord [19], CAN [15], and Pastry [17] aim to overcome this challenge.

The DHT provides storage and retrieval by using a hash function. When a node participates in the DHT system, it is given a range of hash values for which it is responsible. Then the node finds the hash value of the key[1] of the content it has. It then sends [h(key), the content ID, its address] to any node participating in the DHT. The message is forwarded from node to node until it gets to the node responsible for h(key). Once this has been done, the contents can be found by any user; the user needs only to again hash a key to h(key) and ask any node to find the data corresponding with h(key).

In full-text searching, each node stores the posting list for the word(s) it is responsible for. A query involving multiple words requires that the postings for

[1] We call the hash value of x "h(x)".

one or more of the words be sent over the network. For simplicity, this discussion will assume a two-word query. Sending the smaller of the two postings to the node holding the larger posting list is cheaper; the latter node then performs the intersection and ranking and returns the few highest-ranking document identifiers.

According to [13], analysis of 81,000 queries made to a search engine for mit.edu [4] shows that the average query would move 300,000 bytes of postings across the network. Of the queries analyzed, 40% involved just one word, 35% two, and 25% three or more. Google indexes more than 3 billion Web documents [2], and mit.edu has 1.7 million Web pages; scaling to the size of the Web (3 billion pages) suggests that the average query might require 530 MB. If the Internet bandwidth of users is 1 Gbps, and users want to get a reply to their query within 0.5 seconds, for example, the amount of traffic must be less than 0.5 Gb (12.1% of 530 MB).

The normal process of searching for multi-word text in a DHT system is shown schematically in Figure 1 and Table 1-SA. We call this method simple algorithm (SA). The example in the figure and table represents the case of searching for two words, "W1" and "W2". Usually, the transmission from a node to a destination node needs other intermediary nodes; however, in this paper, we omit the intermediary nodes.

In the case of SA, a huge amount of traffic occurs when the node responsible for h(W1) transmits content IDs to the node responsible for h(W2). To reduce this traffic, in the related works we will introduce in Section 2, two main types of measures are taken: using a device for (1) registering contents' keys, or (2) transmitting content IDs. We suggest using a divided bloom filter (DBF), as well as using both devices (1) and (2). First, as regards measure (1), we reduce the amount of traffic in searching for multi-word contents by using a bloom filter ([8], [9]) when a node registers its contents' keys. In addition, as regards measure (2), we reduce the amount of traffic by transmitting the DBF of content IDs in place of the content IDs themselves.

2 Related Work

The bloom filter is used in this paper and in related works in an aim to reduce the amount of traffic in searching for multi-word text in DHT systems. We describe this filter below.

2.1 Bloom filter

A bloom filter is a space-efficient probabilistic data structure used to test whether or not an element is a member of a set. A basic description of a bloom filter and its problem are given in this subsection.

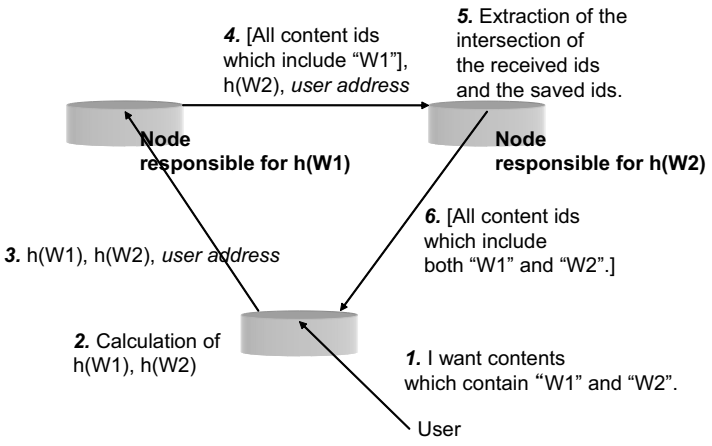

Fig. 1. The process of simple algorithm: normal searching for multi-word text (here, a user want contents which contain the two words "W1" and "W2") on a DHT

Basic description of Bloom Filter Imagine there are set A and set B. To get $A \cap B$ in a simple manner, all the elements of set A are transmitted to the side of set B, and the elements existing in both set A and set B are extracted. At this time, the size of the traffic is the sum of the size of each element in set A. In the method using the bloom filter, set A itself is not transmitted; the bloom filter created by set A is transmitted. The size of the bloom filter is less than the whole size of set A, so the amount of traffic is reduced. The side of set B that received the bloom filter can create s_B satisfying $s_B \supseteq A \cap B$ and $s_B \subseteq B$.

If the test to check whether an element is a member of $A \cap B$ or not to s_B is executed, some false positives (an element that is not a member of $A \cap B$ being returned) occur, but false negatives (an element that is a member of $A \cap B$ being not returned) cannot occur. The false positive rate declines exponentially as the size of bloom filter is increased. Set s_B created by the side of set B is transmitted to the side of set A, and $A \cap B$ is gained.

The execution procedure for the bloom filter is as follows. The idea is to allocate a vector v of m bits, initially all set to 0, and then choose k independent hash functions, $h_1, h_2, ..., h_k$, each with range $1, ..., m$. For each element $a \in A$, the bits at positions $h_1(a), h_2(a), ..., h_k(a)$ in v are set to 1. (A particular bit might be set to 1 multiple times.) Given a query for b, we check the bits at positions $h_1(b), h_2(b), ..., h_k(b)$. If any one of them is 0, certainly b is not in set A. Otherwise, we conjecture that b is in the set, although there is a certain probability that this is incorrect. This is called a "false positive". Parameters k and m should be chosen such that the probability of a false positive (and hence a false hit) is acceptable.

	Simple algorithm (SA)	Transmission fixed-size bloom filter algorithm (TfBFA)	Saving fixed-size bloom filter algorithm (SfBFA)	Saving and transmittion divided bloom filter algorithm (STDBFA)
The contents data N(Wi) contains	Tuple of [h(Wi), content ID, node address]	Same as SA	Tuple of [h(Wi), content ID, node address, fBF]	Tuple of [h(Wi), content ID, node address, DBF]
Execution of UN	Calculation of the DHT hash values h(W1) and h(W2)	Same as SA	Same as SA	Same as SA
Transmission from UN to N(W1)	h(W1), h(W2), UN address	Same as SA	Same as SA	Same as SA
Execution of N(W1)		Creation of a fBF from the saved IDs	Extraction of IDs that have possibilities of containing W2 by using the saved fBFs	Extraction of IDs that have possibilities of containing W2 by using the saved DBFs, and creation of a DBF from the extracted IDs
Transmission from N(W1) to N(W2)	h(W2), saved IDs, UN address	h(W2), the fBF	The rest is same as SA	h(W2), the DBF
Execution of N(W2)	Extraction of the intersection of the received ids and the saved ids	Extraction of IDs that have possibilities of being the constituent element of the fBF N(W2) received, from the IDs registered with h(W2)		Extraction of IDs that have possibilities of being the constituent element of the DBF N(W2) received, from the IDs registered with h(W2)
Transmission from N(W2) to UN	Extracted IDs [Finished]	×		×
Transmission from N(W2) to N(W1)		Extracted IDs		The rest is same as TfBFA
Execution of N(W1)		Extraction of the intersection of the received ids and the saved ids		
Transmission from N(W1) to UN		Extracted IDs [Finished]		

UN : user node N(Wi) : a node responsible for h(Wi)

Table 1. The sequence of searching for multi-word text (here, a user want contents which contain the two words “W1” and “W2”)

The false positive rate (FPR) is a function of k, m, and n, expressed as follows [9].

$$\mathrm{FPR} = (1 - (1 - 1/m)^{kn})^k \tag{1}$$

$$\approx (1 - e^{-kn/m})^k. \tag{2}$$

When $k = \ln 2 \times m/n$, Equation (2) has a minimum value. At that time, FPR is $(1/2)^k$. If the target FPR is set to FPR_{target}, $k = \lfloor \log_{1/2} FPR_{target} \rfloor$. Thus,

$$m = \lfloor \lfloor \log_{1/2} FPR_{target} \rfloor \times n / \ln 2 \rfloor. \tag{3}$$

The salient feature of bloom filters is that there is a clear tradeoff between m and the FPR.

Problem with the Bloom Filter If n (the number of elements of a set) and FPR_{target} are given, the filter bit size m can be minimized by setting parameter k to optimum value. This m value should be shared at the system level. This is because if m is different for different filters, the hash functions differ for checking whether or not a given element is a member of the constituent element of the filter. It is thus necessary to re-calculate the hash value of each element per query. We call the bloom filters for which the sizes are the same "fixed-size bloom filters (fBFs)", and we call the bloom filters for which the sizes are different "variable-size bloom filters (vBFs)". We should use fixed-size BFs in order to avoid to calculate many hash values.

However, if the numbers of sets are different, it is a problem that the filter bit size of fBFs is bigger than that of vBFs on average [18]. This is because the FPR increases exponentially as the number of elements of the set increases under the condition that the filter bit size does not change.

In summary, if we use fixed-size BFs, FPR is higher for the same size of variable-size BF on average. If we use variable-size BFs, calculating hash values takes much time. This comparison is further described in 3.2.

2.2 Reducing the amount of traffic in searching for multi-word in DHT

Several studies have been done to reduce the communication traffic in searching for multi-word text in DHT. Two main developments have come from this research. The first development is a device for registering content keys; the second is a device for transmitting content IDs.

In the first approach, in [11], the set of keywords included in the content was also regarded as a DHT key. The authors created combinations with three words or less, and registered the combinations as well as each word in the DHT. However, the number of combinations increases exponentially as the number of words increases.

In [10], the target for search is a Resource Description Framework [7] (RDF). A system that saves "RDF triples" dispersed in DHT was developed. In this system, the RDF triple itself as well as each element of the RDF triple is registered. Because each RDF triple has only three elements, this method prevented much

extra data storage. However, the method cannot apply to full text searching because the contents have many elements and the extra amount of data storage becomes massive[2].

In [20], a summary of content is registered as DHT keys. Because doing so reduces the number of keys, the amount of traffic in searching for multi-word text was reduced. However, the amount of information was also reduced by summing up content, so this approach cannot apply to the full-text searching we are addressing.

As a second approach, described in [21], [16], and [13], the fixed-size bloom filter is used for transmitting content IDs in searching for multi-word text. By doing so, the amount of traffic was able to reduced without generating any AndSearchData. From here on, we call this method a "transmission fixed-size bloom filter algorithm" (TfBFA).

The specific process using TfBFA is shown in Table 1-TfBFA. In this process, node N(W2), received by the fixed-size bloom filter from node N(W1), transmits the content IDs it extracted to node N(W1) so as to cut off content IDs accidentally included owing to false-positive results.

The advantage of the method using a bloom filter for transmitting content IDs is that there is no AndSearchData; however, a disadvantage of the method is that the reduction rate of the communication traffic is smaller than that in the first approach.

3 Proposed Technique

The related works used a bloom filter for transmitting content IDs, but we also use it for registering the keywords of content. In this section, the problem of the bloom filter and its solution are also described.

3.1 Saving fixed-size bloom filter algorithm (SfBFA)

We developed a device for registering contents' keys. When a node registers its content, it creates a fixed-size bloom filter from all words of the content. Then it registers the filter as well as the hash value of the word to be registered, the content ID, and its address. The specific process for registering contents is as follows.

1. The node calculates the hash values of all words of the content except for word "W1" to be registered.
2. The node creates a fixed-size bloom filter from all hash values it calculated in step (1).
3. The node registers the tuple of h(W1), the content ID, the node address, and the fixed-size bloom filter it created in (2) in the node assigned h(W1).

We call this method a "saving fixed-size bloom filter algorithm" (SfBFA). The process for searching for two-words is shown in Table1-SfBFA.

[2] We call the extra data storage for reducing the amount of traffic "AndSearchData".

Problem of SfBFA As described in subsection 2.1, the optimum filter bit size depends on the number of elements in the set. In this study, the number of elements in the set is the number of words in the content. Because the numbers of words in the content are different, setting the optimum filter bit size becomes a problem.

The k hash functions used in creating the filter should be shared on the DHT system level, so the size of the filter should also be shared on the system level. The filter bit size can be set to be big enough, but the amount of AndSearchData and traffic will be increased. On the contrary, if the filter bit size is set too small, because of the ascension of FPR, the amount of traffic will also be increased.

We do not use variable-size bloom filters because douing so would mean taking too much time to calculate hash values.

3.2 Divided bloom filter (DBF)

We propose divided bloom filters to overcome the problem of bloom filters.

Each filter bit size can thus be maintained by dividing the set into several sets that have the same number of elements and by creating filters from each set. We call filters created by dividing the original set "divided bloom filters" (DBFs). According to Equation 3, m is proportional to n. For this reason, if the FPR of the bloom filter from original set is α, the FPR of each filter of DBFs is also α.

However, the following problem occurs. When an element b is checked as to whether or not it is a constituent element of the DBF, if it is checked through every divided filter and the number of divided filters is GN,

$$\mathrm{FPR} = 1 - (1 - \alpha)^{GN}. \tag{4}$$

If α is sufficiently small, α to the power of more than two can be ignored, so

$$\mathrm{FPR} \fallingdotseq GN \times \alpha. \tag{5}$$

According to this equation, FPR increases as the number of divisions increases.

The solution needs to identify only one filter that can include element b. By this, FPR is equal to α in total. The only filter that can include element b can be identified by using a DHT hash function without creating extra data storage.

When the node divides the set of words in the content, the node calculates the DHT hash value of each word of the content and divides words into groups according to the DHT hash value. In doing so, the system determines the following parameters in advance.

- MN: average number of words each group can include
- Filter bit size and hash functions used to create filters

The specific process to divide the words of content C is as follows. The value that the DHT hash function can return is $1, 2, ..., DN - 1$.

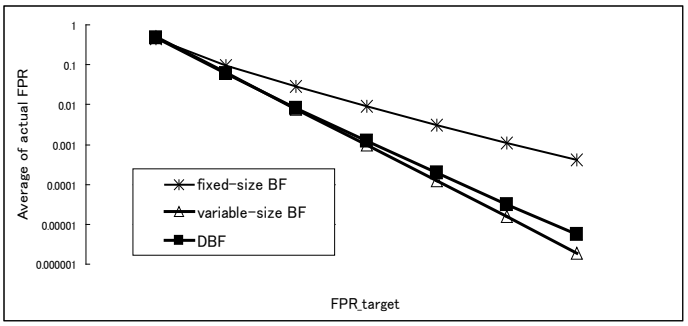

Fig. 2. Average FPR of 1,000 sets (number of elements is from 1 to 1,000)

1. The node calculates the number of groups $GN = \lfloor WN/MN + 0.5 \rfloor$ depending on WN, i.e., the number of the words of content C.
2. The node gives each group $G_i (i = 1, ..., GN)$ the assigned range of value $R(G_i) = [(DN/GN) \times (i-1), (DN/GN) \times i)$.
3. The node extracts a word of content C, considers it as w, and calculates the DHT hash value $h(w)$.
4. If $R(G_j)$ includes $h(w)$, the w is grouped in G_j.
5. The node repeats steps (3) to (4) for all words of content C.

In this method, it is not guaranteed that that each group has the same number of words. However, if the hash function is collision-free, it is assumed that each group has almost the same number of words.

Whether a word b is a member of the words of the content C is determined as follows.

1. The node that received DBF calculates each assigned range of the value $R(G_i)$ of each group G_i according to the number of filters it received.
2. The node calculates the DHT hash value $h(b)$ of word b.
3. The node determines $R(G_j)$ including $h(b)$. At this time, b can be a member of only group G_j.
4. The node judges whether word b can be a constituent element of the filter created by group G_j.

Comparison of fBF, vBF, and DBF Let us compare the following features of fixed-size BFs, variable-size BFs, and DBFs:

1. average FPR in creating filters from several sets that have different number of elements and
2. time complexity where an element is checked as to whether it is a member of the filter.

@

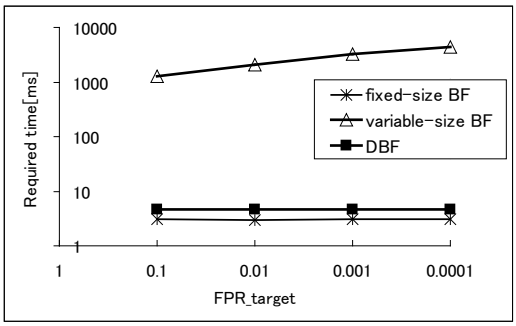

Fig. 3. Required time for checking whether an element is a member of each filter (number of filters is 1,000,000)

1: Figure 2 shows the average FPR of 1,000 sets in each filter method (fixed-size BF, variable-size BF, and DBF). The number of elements of the contents of the sets is from 1 to 1,000. The filter size was determined by FPR_{target}[3]. We changed FPR_{target} from $1/2$ to $1/2^{19}$. MN for the DBFs was set to 100. As FPR_{target} becomes small, we found, the actual FPR of fixed-size BFs becomes much larger than FPR_{target} and that of the DBFs becomes slightly larger than FPR_{target}.

2: Figure 3 shows the simulation result of the required time to check whether an element is a member of a set. We created 1,000,000 filters respectively (fixed-size BF, variable-size BF, and DBF) where the number of elements is 100, and we set $FPR_{target} = 0.1, 0.01, 0.001, \text{and} 0.0001$. We created an element b randomly and measured the required time to determine whether b was a member of each filter. In regards to fixed-size BFs and DBFs, according to Figure 3, the required times do not vary with change in FPR_{target}. In regards to variable-size BF, we recalculated k hash values for each filter. Hence, the required time was very long. In regards to DBFs, the required time was much less than that of variable-size BFs and close to that of fixed-size BFs.

3.3 Saving divided bloom filter algorithm (SDBFA)

We call the method where the node registers a DBF as well as its content ID, its address, and the hash value of the key a "saving divided bloom filter algorithm" (SDBFA).

If this SDBFA is used, the approximate minimum length of the filter satisfying the target FPR can be obtained even if different contents have different numbers of words.

[3] That is, we set the filter size to the size of variable-size BFs whose FPR is FPR_{target}.

3.4 Saving and transmission divided bloom filter algorithm (STDBFA)

An SDBFA can adopt the method using DBF for transmitting content IDs. Doing so realizes the same amount of AndSearchData while decreasing FPR. We call the algorithm-synthesized SDBFA and the method using DBF for transmitting content IDs a "saving and transmission divided bloom filter algorithm" (STDBFA). The process of searching for multi-word text is shown in Table 1-STDBFA.

4 Experiment and Evaluation

Experiments with "simple algorithm" (SA), "transmission fixed-size bloom filter algorithm" (TfBFA), "saving fixed-size bloom filter algorithm" (SfBFA), "saving divided bloom filter algorithm" (SDBFA), and "saving and transmission divided bloom filter algorithm" (STDBFA) were performed. SA does not take any actions for reducing amount of traffic, TfBFA was used in our previous work, and SfBFA, SDBFA, and STDBFA are new methods proposed in this paper.

We measured the average amount of traffic in the five algorithms mentioned above. In addition, we compared the amount of AndSearchData needed for each algorithm.

4.1 Experimental setup

As we described in Section I, the aim of the experiment is to limit the amount of traffic in searching for multi-word text (i.e., 12.1% of SA; 64 MB data from 530 MB data). We prepared 10,000 published papers as contents for the experiment. When we extracted the words of the content, we used the database of vocabulary WordNet [6] and extracted the nouns, verbs, and adjectives included in the content. The virtual user selected two words and searched for contents containing the two words. The general hash function SHA-1 [12] was used as the DHT hash function. Because SHA-1 returns a bit value of 160, the content ID has 160 bits.

The calculation of the amount of traffic generated by TfBFA and STDBFA that use a fixed-size bloom filter or DBF in transmitting content IDs is as follows. In Table 1, in the case of TfBFA and STDBFA, the total amount of traffic is the sum of the amount of traffic node N(W1) transmits to node N(W2) and the amount of traffic node N(W2) transmits to node N(W1). On the other hand, in the case of SA, SfBFA and SDBFA, the amount of traffic is the only amount of traffic node N(W1) transmits to node N(W2).

In this experiment, the average amount of traffic over 1000 trials with simple algorithm was 2.97 KB.

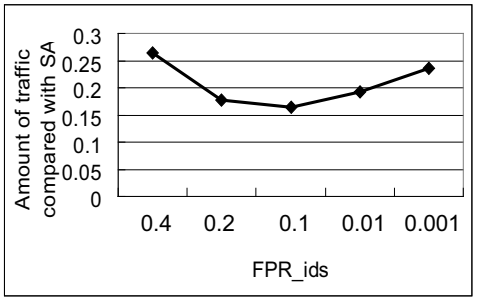

Fig. 4. Average amount of traffic using TfBFA compared with that using SA

4.2 Experimental results

The searches were repeated 1000 times. From here on, we call the FPR_{target} of filters created from content IDs "FPR_{ids}" and call the FPR_{target} of filters created from words included in the contents "FPR_{words}".

In the experiment on TfBFA, we set FPR_{ids} to 0.4, 0.2, 0.1, 0.01, and 0.001. Figure 4 shows the result. The amount of traffic for each of the FPR_{ids}, respectively, was 0.26, 0.17, 0.16, 0.19, and 0.24 compared with that of SA. If the FPR_{ids} is small, the filter bit size that node N(W1) transmits to node N(W2) in Table 1-TfBFA becomes bigger. To the contrary, if the filter bit size is large, the number of content IDs that node N(W2) transmits to node N(W1) becomes larger.

In regards to SfBFA, we set FPR_{words} to 0.4, 0.2, 0.1, 0.01, and 0.001 (Figure 5-Left on the extreme right point and Figure 5-Right on the extreme right point). Figure 5-Left shows the amount of traffic involved in searching for multi-word text, and Figure 5-Right shows the amount of AndSearchData in registering one content to the nodes.

In regards to SDBFA, FPR_{words} was set to the same value as in the experiments with SfBFA, and MN was set to 10, 20, 50, and 100 (Figure 5 except for each extreme right point.) In Figure 5-Left, SDBFA (which uses DBF) can be seen to have reduced the amount of traffic more than SfBFA (which uses a normal bloom filter). As shown in Figure 5-Right, the amount of AndSearchData with the method using a normal bloom filter is not so different from that with the method using DBF. When $FPR_{words} = 0.1$, the goal of 12.1% traffic compared with SA was realized by using DBF. Figure 5-Right shows that the average amount of AndSearchData per content was the same as that with SfBFA. The amount of AndSearchData is the same as that of SDBFA.

In regards to STDBFA, FPR_{words} was set to 0.1 and MN to 10 for registering contents' keys, and FPR_{ids} was set to 0.1 and MN to 2, 5, 10, 20, and 50 for transmission of content IDs (Figure 6.) In Figure 6, the condition $MN = 20$ can be seen to have reduced the amount of traffic the most.

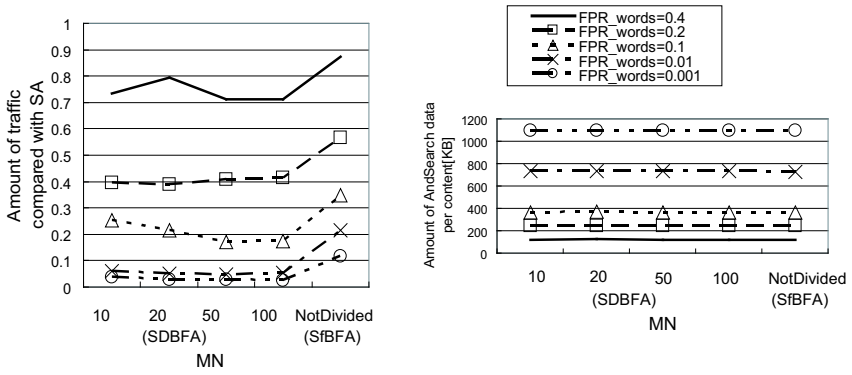

Fig. 5. Left: Average amounts of traffic using SfBFA and SDBFA; Right: Amounts of AndSearchData using SfBFA, SDBFA, and STDBFA

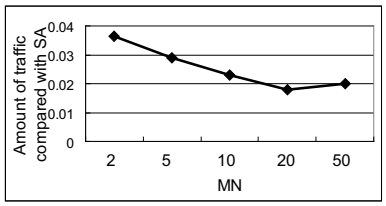

Fig. 6. Amount of traffic using STDBFA

We also examined the effect of changing the number of contents from 1,000 to 10,000 (Figure 7). In regards to TfBFA, the amount of traffic had significant changes. In regards to SDBFA and STDBFA, however, the amount of change in traffic was stably small. Furthermore, the amount of AndSearchData was less than that of SfBFA.

Table 2 is a compilation of the results for all algorithms. The values are the average amount with change in the number of contents from 1,000 to 10,000. In the case of searching for multi-word text, TfBFA used in conventional research used needed 23.7% of the traffic of SA. However, SfBFA (which uses the method of registering fixed-size bloom filters created by all words of the content) reduces the amount of traffic more than TfBFA. In addition, compared to SfBFA, SDBFA and STDBFA (which use the proposed DBF rather than a normal bloom filter) reduce both the amount of traffic and the amount of data storage.

4.3 Discussion

In this work, we set the target as text documents, but we believe that the proposed techniques (SDBFA and STDBFA) can apply to multimedia contents

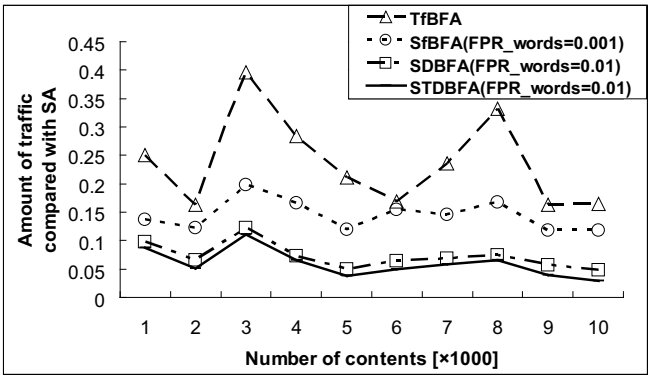

Fig. 7. Amount of traffic with change in the number of contents

	Amount of traffic compared with SA	Amount of data storage per content [KB]
TfBFA	0.237	
SfBFA	0.144	1095
Desired Value	0.121	
SDBFA	0.072	730
STDBFA	0.059	730

Table 2. Comparison of the results for all algorithms

like movies or music. At this time, the keys for DHT are the texts inserted in multimedia contents by languages that describe metadata (like MPEG7 [3]). If mounting metadata into multimedia contents could be done automatically, contents would have much metadata. If a DHT system for these multimedia contents were constructed, the amount of traffic generated in searching for multi-word text would grow larger. However, we believe that our proposed method would also be able to reduce the amount of traffic in such a system.

Some DHT algorithms taking mobility and wireless environments into account have been developed (e.g., M-CAN [14] and Warp [22]). Compared to traditional P2P, characteristics of MP2P include unreliable connection, limited bandwidth, and the constraints of mobile devices. Hence, we believe that our proposed method can better apply to these DHTs.

Note that in the experiments in this work, the virtual user queried random words. However, we should perform experiments by creating a user model from real DHT systems or histories of real search engines.

Furthemore, we only evaluated two-word multiple searching. Three-word multiple searching should be conducted as follows. Let the three words be "W1", "W2", and "W3", and the node responsible for h(W1) be N(W1). In regards to SDBFA and STDBFA, node N(W1) extracts only the content IDs that can include W3 as well as W2; therefore, in these cases, we predict that the amount of traffic would be decreased compared to that of two-word multiple searching.

5 Conclusion

We aimed to reduce the amount of traffic for multi-word searches in DHTs. First, as a device for registering contents' keys, we used a bloom filter created from all words of the content. In this method, some amount of extra data storage for reducing the amount of traffic occurred. We proposed a divided bloom filter (DBF) so as to overcome the limitations of the role of the bloom filter if several sets have different numbers of elements. We used the DBF to reduce the amount of extra data storage as well as the amount of traffic. Second, as a device for transmitting the content IDs, a method by which the node transmits not content IDs themselves, but DBFs of them, was effective in reducing the amount of traffic.

In regards to the saving divided bloom filter algorithm (SDBFA) and the saving and transmission divided bloom filter algorithm (STDBFA) proposed in this paper, we were able to get favorable results for the amount of traffic in searching for multi-word text as well as data storage.

References

1. Gnutella, http://gnutella.wego.com/.
2. Google, http://google.com/.
3. ISO/IEC TR 15938-8:2002: Information technology. multimedia content description interface. part8: Extraction and use of mpeg-7 descriptionscISO/IEC/JTC 1/SC 29, 2002.
4. Massachusetts institute of technology, http://mit.edu/.
5. Napster, http://www.napster.com/.
6. Wordnet, http://wordnet.princeton.edu/.
7. World-wide web consortium: Resource description framework, http://www.w3.org/rdf.
8. Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.
9. A. Broder and M. Mitzenmacher. Network applications of bloom filters: A survey. In *Proceedings of 40th Annual Allerton Conference on Communication, Control, and Computing*, pages 636–646, 2002.
10. Min Cai and Martin Frank. Rdfpeers: a scalable distributed rdf repository based on a structured peer-to-peer network. In *WWW '04: Proceedings of the 13th international conference on World Wide Web*, pages 650–657, New York, NY, USA, 2004. ACM Press.

11. Austin T. Clements, Dan R. K. Ports, and David R. Karger. Arpeggio: Metadata searching and content sharing with chord.
12. D. Eastlake 3rd and P. Jones. US Secure Hash Algorithm 1 (SHA1). RFC 3174, September 2001.
13. J. LI, B. LOO, J. HELLERSTEIN, F. KAASHOEK, D. KARGER, and R. MORRIS. the feasibility of peer-to-peer web indexing and search, 2003.
14. Gang Peng, Shanping Li, Hairong Jin, and Tianchi Ma. M-can: a lookup protocol for mobile peer-to-peer environment. In *ISPAN*, pages 544–550, 2004.
15. Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker. A scalable content-addressable network. In *Proceedings of the ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 161–172, August 2001.
16. Patrick Reynolds and Amin Vahdat. Efficient peer-to-peer keyword searching. In *Middleware*, pages 21–40, 2003.
17. Antony I. T. Rowstron and Peter Druschel. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. In *Symposium on Operating Systems Principles*, pages 188–201, 2001.
18. Michael A. Shepherd, William J. Phillips, and C.-K. Chu. A fixed-size bloom filter for searching textual documents. *Comput. J.*, 32(3):212–219, 1989.
19. Ion Stoica, Robert, David Karger, Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. In *Proceedings of the 2001 ACM SIGCOMM Conference*, pages 149–160, 2001.
20. Chunqiang Tang, Zhichen Xu, and Sandhya Dwarkadas. Peer-to-peer information retrieval using self-organizing semantic overlay networks. In *SIGCOMM*, pages 175–186, 2003.
21. Jiangong Zhang and Torsten Suel. Efficient query evaluation on large textual collections in a peer-to-peer environment. In *Peer-to-Peer Computing*, pages 225–233, 2005.
22. Ben Y. Zhao, Ling Huang, Anthony D. Joseph, and John Kubiatowicz. Rapid mobility via type indirection. In *IPTPS*, pages 64–74, 2004.