A Novel Content Distribution Mechanism in DHT Networks*

Quanqing Xu[1], Heng Tao Shen[2], Bin Cui[1], Xiaoxiao Hou[3], and Yafei Dai[1]

[1] State Key Lab. for Adv. Opt. Commun. Syst. & Networks, Peking University, 100871 Beijing, China
{xqq,dyf}@net.pku.edu.cn, bin.cui@pku.edu.cn
[2] School of ITEE, The University of Queensland, Brisbane QLD 4072, Australia
shenht@itee.uq.edu.au
[3] Department of Computer Science and Technology, Stony Brook University, U.S.A.
xhou@cs.sunysb.edu

Abstract. DHT (Distributed Hash Table) is a structured overlay network that is widely utilized in P2P systems. Existing content distribution approaches do not completely exploit features of DHT and incur heavy network bandwidth consumption. This paper analyzes existing content distribution approaches including synchronous content distribution method in eMule based on DHT overlay networks and points out that their network loads are too heavy. We propose a novel content distribution algorithm: asynchronous distribution, in DHT networks. Compared with traditional distribution approaches, it is more effective and scalable with lower network load. We apply the techniques of vector space model and search frequency to the asynchronous distribution algorithm, which effectively improves search hit ratio and reduces network load. Simulation results based on real data from Maze system show that this approach has low network overhead and publishing cost, high search and download hit ratio.

Keywords: DHT, Content Distribution, Vector Space Model, Search Frequency.

1 Introduction

DHT, e.g., Chord [1] and Pastry [2], is a structured overlay network that is widely utilized in P2P systems, e.g., file-sharing systems like eMule[1], BitTorrent[2], and AmazingStore[3]. These file-sharing systems employ Kademlia [3] as their DHT

* This research has been supported by the National Grand Fundamental Research 973 program of China under Grant No.2004CB318204, Australian research grant (ARC DP0773483) and National Natural Science foundation of China under Grant No.60873051.

[1] http://www.emule-project.net
[2] http://www.bittorrent.com/
[3] http://amazingstore.grids.cn

L. Fratta et al. (Eds.): NETWORKING 2009, LNCS 5550, pp. 742–755, 2009.

infrastructure to improve performance. However, existing content distribution approaches, e.g., synchronous distribution algorithm in eMule, do not adequately exploit features of DHT and wastes precious network bandwidth. DHT maps object keys to overlay nodes and offers a lookup primitive to send a message to the node[4] responsible for a key. Overlay nodes maintain routing states to route messages towards the nodes responsible for their destination keys, which brings good locating performance. However, the retrieval ability of DHT is very limited and DHT does not have the requisite ability to rank search results because DHT only provides the exact match function.

We call *content distribution (publishing)*: it is a procedure that contents are provided to other people by given ways, which aims to implement resource-sharing or resource-distributing. Distributed objects are sharing-files of various formats, documents, etc. In this paper, we propose a novel Asynchronous Content Distribution approach (ACD). We analyze existing distribution approaches including synchronous publishing method in DHT networks and points out that their network load balances are heavy. We propose a novel asynchronous content distribution algorithm in DHT networks. Compared with traditional distribution approaches, it is more effective and scalable with lower network load. We apply the techniques of vector space model and search frequency to the asynchronous distribution algorithm, which improves search hit ratio and reduces network load. The main contributions of this paper are threefold:

- We present an asynchronous content distribution algorithm under DHT environments, which solves existing issues in the synchronous distribution algorithm;
- We utilize vector space model and user relevance feedback from traditional information retrieval domain to improve search hit ratio and decrease publishing cost in DHT networks;
- We confirm the effectiveness and efficiency of our methods by conducting an extensive performance measured by network overhead, publishing cost, search and download successful ratio.

The remainder of this paper is organized as follows: Section 2 presents the problem formulation and reviews related work. We design an asynchronous distribution algorithm in DHT networks in Section 3 and propose optimization strategy in Section 4. Section 5 reports the experimental results. Section 6 concludes this paper and discusses future work.

2 Preliminaries

2.1 Semantic Operations and Applications of DHTs

After DHTs appeared, there are increasing applications based on structured P2P, and many mature DHTs are utilized in real systems. Thus, it is necessary to analyze functions of a DHT and give formal definitions. From the standpoint

[4] In this paper we use the terms “node” and “peer” interchangeably.

of function, a DHT provides a hash function, publishes a content based on its key and locates the content through its key in DHT networks. From the standpoint of topology, a DHT provides a distributed routing algorithm that disperses the routing function of overlay to all the peers: each peer can lookup other peers and forward routing messages. A general DHT provides the following semantic operations:

1) **Put(**Key**,** $Value$**)** Publishes a pair (Key, $Value$) into DHT networks;
2) **Get(**Key**)** Returns the $Value$ of the Key;
3) **Lookup(**Key**)** Provides general access to the node maintaining the Key;
4) **Routing(**Key**)** Provides general access to the node responsible for the Key, and to each node along the routing path.

A DHT provides two basic operations [4]: Put and Get. In fact, Put and Get are functions of DHT application layer, while Lookup and Routing are functions of DHT overlay layer. Put/Get and more complicated operations can be implemented through Lookup and Routing operations. Therefore, Lookup/Routing are basic operations of a DHT. OpenDHT [5] provides interfaces including Lookup, Routing and Put/Get for upper applications.

2.2 Traditional Content Distribution Approaches in DHT Networks

A content distribution (publishing) and retrieval algorithm under DHT environments includes two phases: 1) publishing phase, it is a procedure that a peer publishes its local information to DHT networks by a given policy; 2) retrieving phase, it is a procedure that a peer employs a keyword to retrieve relevant results. After the publishing phase, there are a large number of contents in DHT networks. Peers can retrieve relevant contents in the retrieving phase. Therefore, different publishing policies determine index mechanisms and retrieval approaches used in DHT networks. Basically, every content distribution method in DHT networks includes these two phrases. Note that they are independent in logic but parallel in real systems. Generally, there are three interface functions:

1) **Hash(Object)** Calculates the hash value of an object;
2) **Publish(**$\langle Key, Value\rangle$**)** Publishes a pair $\langle Key, Value\rangle$;
3) **Segment(Doc)** Segments a document.

In traditional information retrieval (IR) field, a retrieval procedure is that a relevant document list is achieved and ranked for a given query. Here we utilize concepts in IR fields. A retrieval procedure in DHT networks is that a document list is retrieved through a query that includes a term or several terms. In general, a content publishing and retrieving algorithm is classified into three categories according to publishing mode.

Term-based Content Publishing Approach. We present a schematic diagram of term-based content publishing approach reads as shown in Figure 1. $Node_1$ segments its documents into terms and publishes the terms into DHT networks. $Term_1$ and $Term_2$ are maintained respectively by $Node_2$ and $Node_3$ in Figure 1. The merits of this algorithm are: 1) each term of each document is

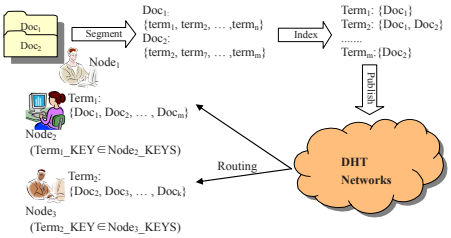


Fig. 1. Term-based Publishing Approach

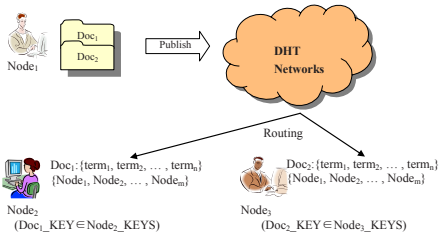


Fig. 2. Doc-based Publishing Approach

published; 2) a fully distributed inverted index is created; 3) a peer can easily achieve the global information of a term that it is responsible for and calculates the IDF (Inverse Document Frequency) value of this term; 4) each peer can quickly retrieve and rank search results for a given query term in DHT networks.

Document-based Content Publishing Approach. We give a schematic diagram of this approach is shown in Figure 2. $Node_1$ publishes its documents into DHT networks. Doc_1 and Doc_2 are maintained respectively by $Node_2$ and $Node_3$ in Figure 2. The advantage of this algorithm: each peer maintaining a document can know all the terms of the document so that the similarity between a term and a document can be calculated when a search is conducted. However, its disadvantage is that it can not use DHT to perform a search for a term. In fact, this approach is utilized in the DHT of BitTorrent: peers downloading the same file resource publish their addresses so that a downloading peer can find the peers including the file, which is helpful to improve multi-peers download efficiency.

Hybrid Content Publishing Approach. Analyzing merits and demerits of the above two approaches, Tang et al. [6] in eSearch presented a hybrid content publishing algorithm: executing a two-phase protocol to publish a content (document). In the first phase, it utilizes the publishing algorithm based on term objects and employs DHT routing to locate the peers responsible for top terms in the document. In the second phase, it uses the publishing algorithm based on document objects. The initiator can build the term list for the document and multicast the term list or the document itself. This publishing algorithm has two advantages: 1) The returned results can be ranked based on the similarity between a query and a document; 2) It can support complex queries. With document in hand, eSearch can search exact matches for quoted text, provide sentence context for matching terms, and support a "cached documents" features to decrease network load. However, this publishing algorithm is proposed as a full text retrieval approach in DHT networks. Each term is published with lots of documents, which causes very huge network load. Therefore, this algorithm is not suitable for fully opening P2P systems.

The first two publishing approaches are commonly used under DHT environments. The third approach aims to utilize DHT as an infrastructure for distributed full text retrieval where the published object is a document or web page, which

often includes hundreds of terms. The third one is executed among relatively stable peers, e.g., all the machines in a lab or PlanetLab[5], and very large bandwidth consumption is generated. These algorithms do not consider an opening P2P system, which needs to face high node churn and limited bandwidth.

2.3 Other Related Work

Coral [7] is a free P2P content distribution network designed to mirror web content. One of Coral's key goals is to avoid ever creating hot spots that might prevent peers from running the Coral because of concerning about overload. Coral uses an indexing abstraction called a distributed sloppy hash table (DSHT) to achieve this goal. Self-organizing clusters of nodes are created by DSHTs to fetch information from each other, which can avoid communicating with more distant or heavily-loaded servers. Yang et al. [8] proposed a Chord-based distributed architecture for content-based publish/subscribe services. This infrastructure is a scalable platform and can simultaneously support many publish/subscribe schemas. Steiner et al. [9] studied the content management process implemented of KAD as implemented in aMule[6]. CoBlitz [10] is a scalable Web-based distribution system for large files, which distributes large files without requiring any modifications to standard Web servers and clients, since all the necessary support is located on the CoDeeN content distribution network.

3 Asynchronous Content Distribution Algorithm

3.1 Requirement Analysis of Search in P2P File-Sharing Systems

In this subsection, we briefly analyze existing problems about information retrieval that need to be considered in P2P file-sharing systems. The published and indexed objects are files shared by users in P2P file-sharing systems. These file objects have particular characteristics.

Each file has a unique file name used by a specific user. However, the same file often has many different file names applied by different users due to:

1) Different users have different habits of naming sharing files: some users like making file names normal, while others like making file names arbitrary;
2) Some users often insert many hot query terms into an ordinary file for attracting other users to download it.

Thus, each file often has different file names, which increases the retrieval difficulty in P2P file-sharing systems.

Generally speaking, there are two different requirements when initializing a query:

1) A relevant file list is achieved through a keyword used by a user. The user wants to retrieve as many files as possible and ranks these files so that the user can select the desired files and download these files quickly;

[5] http://www.planet-lab.org
[6] http://www.amule.org/

2) The user uses the key of a file object to retrieve as many users that own this file object as possible so that he/she can download it from many users.

It is easy to meet requirement 2), while it is hard to satisfy requirement 1) in DHT systems. The above-mentioned three traditional index structures can not satisfy the two requirements. Only the synchronous publishing algorithm in eMule can meet them.

3.2 Asynchronous Publishing

There are two kinds of objects that need to be published: file object and file name (term) object in P2P file-sharing systems. The owner of a file object is responsible for publishing the file object and its file name object in eMule. We define *synchronous publishing*, as the method where these two objects are published by the same peer. The advantage of synchronous publishing is that both a file object and its file name object are published by the same logic role. Therefore, its system architecture is relatively easy to be understood and implemented. Moreover, synchronous publishing is relatively good in performance, which is seen from that eMule is widely popular.

However, synchronous publishing does not consider the following problems:

1) Although the same file has different file names applied by different users, most of important terms are the same. However, these terms are published repeatedly by different users. Therefore, this method brings too many redundant messages;
2) Some file names have many terms. However, some of these terms are not relevant, e.g., stop words, which are not necessary to be published. A user can not publish terms according to the importance of terms because the user do not know the file names of the same file applied by other users and can not determine which terms are important to this file;
3) There is no global information for a file object. A peer responsible for a term object can not determine the relevance between this term and its corresponding files.

We comprehensively analyze these three defects and draw a conclusion that the owner of a file object published the terms of this file and results in short of useful information when publishing these terms. If the file names of a file object can be obtained, and the terms of the file object are published, this problem can be solved. Therefore, we propose an asynchronous publishing algorithm to solve the before-mentioned issues. The basic idea is that the owner of a file object publishes this file, and the maintainer of this file integrates all the file names of this file. The maintainer segments these file names to get the relevant terms, and publishes these terms in DHT networks. We introduce the details of this algorithm in the next subsection.

3.3 Specific Algorithms

The basic idea of the asynchronous publishing algorithm is that the owner of a file only publishes this file itself and the maintainer of this file publishes those

Algorithm 1. Content Asynchronous Publishing Algorithm

```
   // Publishing File objects
1  for File ∈ Node1 do
2      File_Key=Hash(File);
3      File_Value={Node_Addr,File_Name};
4      Publish(⟨File_Key, File_Value⟩);
   // Merging File objects
5  Node2 receives the published File (File_Key ∈ Node2_Keys) object from
   nodes;
   /* A file list includes:(a) Node_Addrs, which is used to find much
      more download links; (b) File_Names, which records file names
      used by different peers                                        */
6  Node2 merges the values of File and maintains them;
   // Publishing Term objects
7  Node2 segments the File_Name list of the files maintained by it and get the
   Terms;
8  for Term ∈ Terms do
9      Key=Hash(Term);
10     Make File_Key and File_Name be the value of Term;
11     Publish(⟨Key, Value⟩);
   // Merging Term objects
12 Node3 receives the published Term object from nodes in DHT networks;
13 Node3 maintains a file object list for each received term object;
   // for all the files in DHT networks
14 All the nodes repeat the above steps;
```

terms in the names of this file. We present the asynchronous publishing algorithm as shown in Algorithm 1, the search and download algorithm as shown in Algorithm 2.

The asynchronous publishing algorithm's schematic diagram is shown in Figure 3. The algorithm integrates all the different file names of the same file, segments these file names into terms and publishes these terms. This algorithm can decrease redundant messages to some extent and integrate the file names

Algorithm 2. Content Search and Download Algorithm

```
   Input: Term
   // Node1 uses a term to perform a search
1  Term_Key=Hash(Term);// Term_Key ∈ Node2_Keys
   // routes to Node2 through DHT
2  Node2 searches its local information based on Term_Key;
3  Node2 returns the File object list of Term_Key to Node1;
4  Node1 selects File1 from the File object list to download;
5  Node3=Lookup(File1_Key) by Node1;
6  Node3 returns all the nodes including File1;
7  Node1 downloads File1 from those nodes;
```

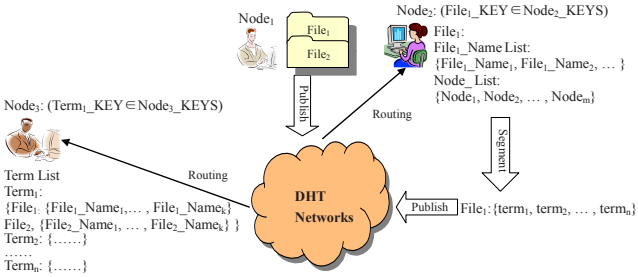


Fig. 3. The Schematic Diagram of Asynchronous Publishing Algorithm

of files to distinguish fake files because users may name them with fake names. Most of all, this algorithm can integrate the global information of a specific file in the same peer, it can also integrate the global information of a specific term in the same peer. We can use this global information to optimize the publishing algorithm. Although a term is published with full file name of a file in this algorithm, the full file name is much shorter than a document. Therefore, compared with the full text retrieval in DHT networks, this algorithm will not bring too much network load.

4 Optimizations for Asynchronous Publishing

Generally, different term in a file name has different importance to the same file object. Therefore, we can improve the asynchronous publishing algorithm based on the importance of terms in a file object when publishing these terms. Important terms are published with high frequency, while those unimportant terms are published with low frequency or are not even published, which can dramatically reduce bandwidth consumption and guarantee search performance. Moreover, the same term has different weights in different files. Thus, we can utilize some weight schemes to calculate a term's weight in a file name, which can rank search results and improve users' experience. In this section, we present two optimizations for asynchronous publishing based on weight schemes of vector space model (VSM) [11], which is widely used in information retrieval.

4.1 Optimization I: Term Frequency

TF*IDF (Term Frequency - Inverse Document Frequency) is a weight rule often used in information retrieval. A TF*IDF weight is a statistical measure used to evaluate how important a term is to a document in a corpus. The importance increases proportionally to the number of times (TF) that a term appears in a given document but is offset by the document frequency (DF) of the term in the corpus. In unstructured P2P networks, however, it is impossible for us to calculate the global IDF values since the calculation involves counting the document frequency of terms. On the contrary, it is easy to calculate the global IDF values

since the same term is maintained by the same peer in DHT networks. Therefore, an effectively calculating formula reads as follows: $IDF_{term} = \log(N/DF_{term})$, where N is a maximum unsigned integer.

Each file may have different file names for different users in P2P systems. If these file names are integrated into a "document": using the weight of a term in this document to indicate the importance of this term in this file object. The key of this approach is to get the TF and IDF values of this term. In the synchronous publishing method in eMule and the publishing methods in Section 2.2, the two values can not be achieved simultaneously. However, both of them can be obtained in the asynchronous publishing method. The concrete approach reads as follows:

In line 7 of the asynchronous publishing algorithm, when $Node_2$ segments the $File\_Name$ list of all the files maintained by it, it can count terms' frequency in file names. These terms with their TFs are published. $Node_3$ receives the published $Term$ object from nodes in DHT networks, inserts it into the local $Term$ list and counts *file (document) frequency* (DF) of this term. $Node_3$ returns this term's IDF to $Node_2$. Therefore, both $Node_2$ maintaining the file object and $Node_3$ maintaining the term object have a TF for this term in a given file and the IDF of this term. These two nodes can utilize the two values to do different things.

$Node_3$ can use TF*IDF to measure the relevance of each term in a file object so that search results can be ranked according to this relevance when a user performs a search for a term. Moreover, if a user wants to search a file, he/she can use a term considered as the most relevant term to that file. In other words, the more relevant to a file a term is, the more possible it is utilized to find this file by a user. In addition, a file has many terms, the weights of some of them are small and are not used to find this file by a user. Thus, $Node_2$ can decide the publishing frequency of a term maintained by it according to its TF*IDF. If a term has a high relevance, it will be published with normal publishing frequency. If not, it will be published with a lower publishing frequency, which can decrease network load.

This approach seems to increase a little network load: 1) The length of a message is increased because of adding TF values in the phase of publishing terms; 2) $Node_3$ needs to respond with IDF values to $Node_2$, which increases message numbers. However, both TF and IDF are so short that they are negligible and an IDF can be inserted into a response message of $Node_3$. Therefore, they have marginal influence on network load. In addition, network load can be reduced because publishing frequencies of irrelevant terms are decreased.

4.2 Optimization II: Search Frequency

To some extent, the TFIDF rule can distinguish the importance of each term, but it can not totally reflect this importance. Therefore, we may better distinguish the importance of each term if we consider users' search behaviors. It is easy to perform this in P2P file-sharing systems. If a user wants to search a file, he/she uses a keyword, which is considered by him/her to be the most relevant

term to this file. In other words, a term is published to make users easily find corresponding files. A term often used by users for retrieval should be important. Thus, we can use search frequency to measure the relevance between a term and a file.

In line 5 of Algorithm 2, $Node_1$ may send the used keyword in line 1 to $Node_3$ when performing a lookup to find all the nodes publishing $File\_Key$ via $Node_3$ for downloading files. Therefore, $Node_3$ can know keywords used by users when these users download the files maintained by $Node_3$. $Node_3$ can locally save these keywords' frequencies for being used to retrieve. This frequency is called as *search frequency* (SF). $Node_3$ can utilize *search frequency* to optimize the proposed asynchronous publishing approach.

Note that there are no users' feedbacks at the beginning of file-sharing. Thus, this approach may be employed with TF*IDF. TF and SF are similar, which are used to measure the relevance between a term and a file. However, SF is a result of users' feedbacks and is much more important than TF. Therefore, the weight of SF is higher than that of TF when they are both utilized. This approach only adds a keyword before users find much more files and download them. This keyword is so short that the increment of network load may be negligible.

5 Experiments

5.1 Data Sets

The data sets come from shared files from 28/10/2007 to 09/11/2007 in a real P2P system: Maze[7]. There are 7,565 active users and 30,001,293 files totally. First of all, we preprocess the shared files, e.g., removing shared files in a system disk (e.g., C:\), WINDOWS installation directory (e.g., C:\WINDOWS) and programs installation directory (e.g., C:\Program Files), which are shared by free riders. Then, we choose 6,144 active users, and the files they shared are 5,543,293. To simulate users' feedbacks, we utilize the search log in 10/2007 to extract the data with high search frequency. We extract real users' queries from a real log from 28/10/2007 to 09/11/2007 in Maze system to be employed as queries in our simulations. In addition, We utilize online, offline and enter time data of active users from logs of Maze system to simulate peer churns of P2P systems, where time interval Δt is 3˜5 minutes. Kademlia [3] is utilized as our DHT infrastructure. Table 1 summarizes experimental parameters. Notations in the experiments are shown in Table 2.

5.2 Experimental Results

Experiment 1: Comparison of Synchronous and Asynchronous Publishing Algorithms. In this experiment, we compare existing synchronous publishing algorithm in eMule with the proposed asynchronous one. File objects need to be published in both algorithms. Therefore, we only concern the publishing

[7] A P2P file sharing system with millions of registered users, http://maze.pku.edu.cn

Table 1. Parameter Setting

Parameter	Setting
# of nodes	64,128,. . . ,1024,2048,3072,. . . ,6144
Publishing ratio	0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0
# of queries	18,846
Updating (h)	0.5,1.0,1.5,2.0,2.5,3.0,3.5,4.0,4.5,5.0
# of copies	1,2,3,4,5,6

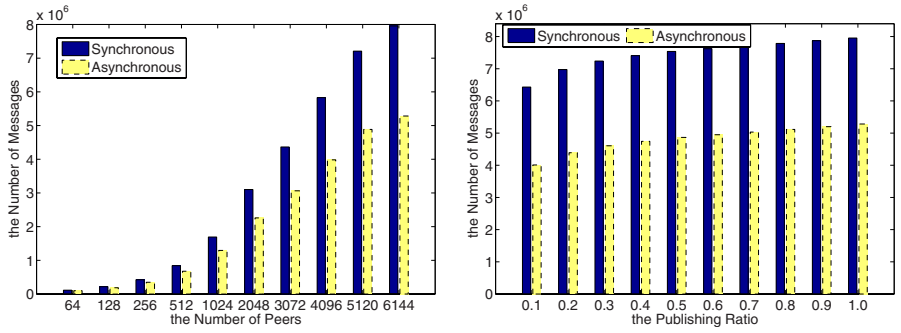
Table 2. Notations

Symbol	Description
N_s	Network Size
P_r	Publishing Ratio
U_t	Index Update Time
U_c	Index Update Copies
N_m	# of Messages

procedure about file name objects. The procedure is that term objects are published as file names are segmented. Synchronous and asynchronous publishing algorithms are only different in publishing mode. They do not affect final publishing and retrieval results because the same file and term objects are maintained by the same peer according to DHT's principle.

Compared with the synchronous publishing approach in eMule, the asynchronous one can reduce more messages as there are increasing peers in DHT networks, which is shown in Figure 4(a). The reason for this is that the asynchronous publishing approach concerns that the same file has many copies in P2P file-sharing systems and most of them have the same important keywords. For the synchronous publishing, these keywords are repeatedly published by different users so that a lot of redundant messages are produced.

In addition, we also explore changes of messages' quantity produced by these two algorithms as the publishing ratio varies with the popular degree of keywords. The smaller the publishing ratio of keywords is, the bigger the percentage of messages deceased by the asynchronous approach is, as shown in Figure 4(b). There is the same reason as explained about Figure 4(a). Moreover, the more popular keywords are, the more files including these keywords are. Therefore, the asynchronous publishing algorithm produces less messages than the synchronous one. Compared with the synchronous method, the asynchronous method can decrease bandwidth consumption, achieve better publishing performance and scalability.



(a) Performance with Different N_s (P_r=1.0) (b) Performance with Different P_r (N_s=6144)

Fig. 4. Comparison of Synchronous and Asynchronous Publishing Algorithms

Experiment 2: Comparison of Asynchronous Publishing and Optimized Counterparts. This experiment mainly explores three kinds of algorithms:

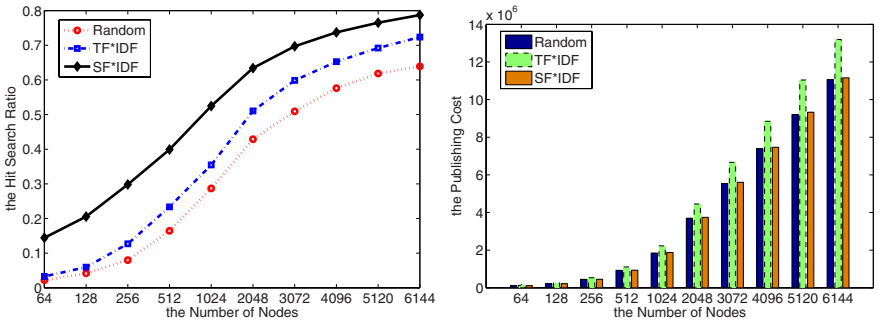
1) *Random*: it is the randomly asynchronous publishing algorithm, where the published keywords are randomly selected in the asynchronous publishing algorithm;
2) *TF*IDF*: it is the asynchronous publishing algorithm with optimization I: Term Frequency, where the published keywords are selected according to the TFIDF rule;
3) *SF*IDF*: it is the asynchronous publishing algorithm with optimization II: Search Frequency, where the published keywords are selected according to the TFIDF rule with search frequency.

We utilize search hit (successful) ratio and publishing cost to evaluate these three algorithms. Search hit ratio is a percentage that indicates the proportional amount of the number of successful queries to the number of all the queries. Publishing cost (PC) is the bandwidth consumption when publishing file name objects in the procedure of asynchronous publishing, which is defined by:

$$PC = \sum_{peer} \sum_{term} (\|term\| + 16 + 20 \times PeerNum_{term}) \tag{1}$$

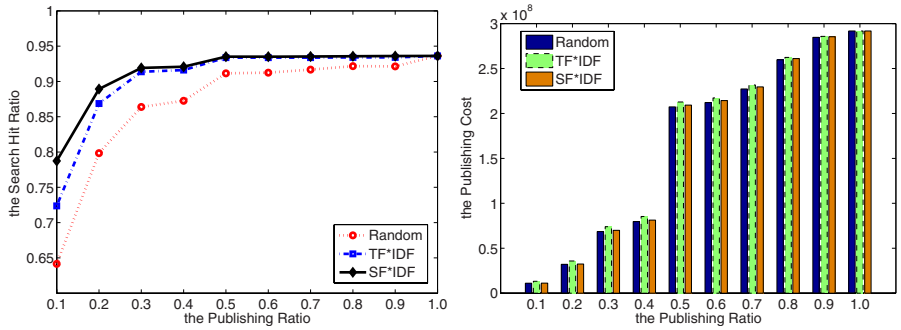
where $\|term\|$ represents the length of $term$, a file object is represented by MD5 (16 bytes), each peer's ID is represented by 160 bits (20 bytes) according to Kademlia.

For the test data sets, *SF*IDF* achieves the best performance among the three algorithms as shown in Figure 5(a) and Figure 5(b). *TF*IDF* is better than *Random* in search hit ratio, but it increases the publishing cost, which leads to heavier load in each peer. It is because when a proportion of popular keywords are published in DHT networks, their publishing frequencies are higher than those of common keywords. Based on the TFIDF rank rule, *TF*IDF* selects keywords from file names. The published keywords' records include more peers and files than those in *Random* and *SF*IDF*. *SF*IDF* is directed by search frequency, so it is better



(a) Search Hit Ratio (b) Publishing Cost

Fig. 5. Comparison of Publishing Performance with Different N_s (P_r=0.1)



(a) Search Hit Ratio (b) Publishing Cost

Fig. 6. Comparison of Publishing Performance with Different P_r (N_s=6144)

than *TF*IDF*, and is similar to *Random* in publishing cost. However, *SF*IDF* is better than *TF*IDF* and much better than *Random* in search hit ratio. From the results in Figure 6(a) and Figure 6(b), these algorithms based on the rank rule is much better than *Random* in search hit ratio as the publishing ratio is smaller. Moreover, *SF*IDF* is better than the other algorithms in the overall performance as the publishing ratio is smaller. The reasons for this are similar to those of Figure 5(a) and Figure 5(b), where they are not explained in detail.

6 Conclusions and Future Work

In this paper, we firstly analyzed existing publishing algorithms including synchronous publishing method in eMule and points out that their network loads are too heavy. The proposed method is more effective and scalable with lower network load than traditional publishing algorithms. We applied two optimization techniques to the proposed algorithm, which can effectively improve search hit ratio and reduce network load. Simulations based on real data from Maze system show that this approach has low network overhead and publishing cost, high search and download hit ratio.

Future work includes: Integrating a real P2P system with the proposed approach in this paper and applying this approach to full text retrieval under DHT environments, etc. Lunar[8] is an ongoing project in our research team, which aims to provide a general middleware for P2P systems, such as P2P file-sharing systems, P2P storage systems. There are several problems that need urgent solutions: how to publish contents to decrease bandwidth under DHT environments? how to design a better retrieval algorithm to rank returned results? how to design a better index mechanism to improve the availability of systems. Based on Lunar, a new Maze system: LMaze is developed. Although the proposed approach in this paper is designed for information retrieval in P2P file-sharing systems, it is also suitable for full text retrieval under DHT environments, such as Overcite [12].

[8] http://maze.pku.edu.cn/lunar.htm

References

1. Stoica, I., Morris, R., Karger, D.R., Kaashoek, M.F., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for internet applications. In: SIGCOMM, pp. 149–160 (2001)
2. Rowstron, A.I.T., Druschel, P.: Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In: Guerraoui, R. (ed.) Middleware 2001. LNCS, vol. 2218, pp. 329–350. Springer, Heidelberg (2001)
3. Maymounkov, P., Mazières, D.: Kademlia: A peer-to-peer information system based on the xor metric. In: Druschel, P., Kaashoek, M.F., Rowstron, A. (eds.) IPTPS 2002. LNCS, vol. 2429, pp. 53–65. Springer, Heidelberg (2002)
4. Dabek, F., Zhao, B.Y., Druschel, P., Kubiatowicz, J., Stoica, I.: Towards a common api for structured peer-to-peer overlays. In: Kaashoek, M.F., Stoica, I. (eds.) IPTPS 2003. LNCS, vol. 2735, pp. 33–44. Springer, Heidelberg (2003)
5. Rhea, S.C., Godfrey, B., Karp, B., Kubiatowicz, J., Ratnasamy, S., Shenker, S., Stoica, I., Yu, H.: Opendht: a public dht service and its uses. In: Guérin, R., Govindan, R., Minshall, G. (eds.) SIGCOMM, pp. 73–84. ACM, New York (2005)
6. Tang, C., Dwarkadas, S.: Hybrid global-local indexing for efficient peer-to-peer information retrieval. In: NSDI, pp. 211–224 (2004)
7. Freedman, M.J., Freudenthal, E., Mazières, D.: Democratizing content publication with coral. In: NSDI, pp. 239–252 (2004)
8. Yang, X., Hu, Y.: A dht-based infrastructure for content-based publish/subscribe services. In: Hauswirth, M., Wierzbicki, A., Wehrle, K., Montresor, A., Shahmehri, N. (eds.) Peer-to-Peer Computing, pp. 185–192. IEEE Computer Society, Los Alamitos (2007)
9. Steiner, M., Carra, D., Biersack, E.W.: Faster content access in kad. In: Wehrle, K., Kellerer, W., Singhal, S.K., Steinmetz, R. (eds.) Peer-to-Peer Computing, pp. 195–204. IEEE Computer Society, Los Alamitos (2008)
10. Park, K., Pai, V.S.: Scale and performance in the coblitz large-file distribution service. In: NSDI (2006)
11. Salton, G., Wong, A., Yang, C.S.: A vector space model for automatic indexing. Commun. ACM 18(11), 613–620 (1975)
12. Stribling, J., Li, J., Councill, I.G., Kaashoek, M.F., Morris, R.: Overcite: A distributed, cooperative citeseer. In: NSDI (2006)