A Novel Method for Content Consistency and Efficient Full-text Search for P2P Content Sharing Systems

Hideki Mima
University of Tokyo
7-3-1 Hongo, Bunkyo-ku, Tokyo, Japan
mima@biz-model.t.u-tokyo.ac.jp

Hideto Tomabechi
Cognitive Research Lab
7-8-25 Roppongi, Minato-ku, Tokyo, Japan
Hideto_Tomabechi@crl.co.jp

Abstract

A problem associated with current P2P (peer-to-peer) systems is that the consistency between copied contents is not guaranteed. Additionally, the limitation of full-text search capability in most of the popular P2P systems hinders the scalability of P2P-based content sharing systems. We proposed a new P2P content sharing system in which the consistency of contents in the network is maintained after updates or modifications have been made to the contents. Links to the downloaded contents are maintained on a server. As a result, the updates and modifications to the contents can be instantly detected and hence get reflected in future P2P downloads. Natural language processing including morphological analysis is performed distributedly by the P2P clients and the update of the inverted index on the server is conducted concurrently to provide an efficient full-text search. The scheme and a preliminary experimental result have been mentioned

1 Introduction

P2P content sharing systems can distribute large amounts of contents with limited resources. By utilizing this exceptional feature, the P2P content sharing model is expected to be one of the major means for exchanging contents.

However, the presently available P2P content sharing systems are mainly used to illegally copy movies and music contents. In some cases, the service providers are accused of such illegal data exchange.

We have recognized that the following technical problems may result in the above mentioned misuse of P2P.

First, the presently available commercial P2P content sharing systems do not provide sufficient functions to track the exchange of contents among users. Due to this, service providers cannot monitor the illegal exchange or tampering of shared contents among users.

Second, the presently available commercial P2P content sharing systems only provide simple search functions, such as keyword search; therefore, they are unsuitable for contents that are either frequently updated or have text. In practice, the current P2P content sharing systems are mainly used to only share movies and music contents because these are not frequently updated. The development of an appropriate search method for the P2P content sharing system is required in order to apply them to search text contents and the latest version of contents.

In order to solve these technical problems, we are developing a content consistency maintenance method and an information search technique for P2P content sharing systems. Our content consistency maintenance method consists of a technique that prevents the tampering of contents and a method that maintains consistency between the following:

1. how users exchange contents on a P2P contents sharing system and
2. how the service provider recognizes the exchange of contents.

Finally, we aim to standardize the result of previous research [10].

In order to handle the updates of contents, the P2P content sharing system that we are developing maintains digital signs for each version of the content. Our system uses a download protocol based on asymmetric key encryption to maintain content consistency. In order to obtain the latest version of contents, even for updated contents, this method employs links to the original and the downloaded contents. These links are managed on a central server.

In order to efficiently implement a full-text search, clients connected to our system perform morphological analysis and summarization of the text to generate text information that is necessary for building a reverse index on a central server. The text information is stored on a central server when the content is updated. To reduce the load of full-text search, the search results are cached on clients. By these techniques, we can distribute the load of natural language processing among clients and rapidly search text contents with content updates.

In this paper, we briefly describe the P2P content sharing system that we are developing and the techniques used in it, namely, a content consistency maintenance method and a full-text search method. We also report the result of a preliminary experiment on load balancing of full-text search by our technique.

This paper is structured as follows: Section 2 describes related work. Section 3 briefly describes the

P2P content sharing system that we are developing. Sections 4 and 5 describe techniques for content consistency maintenance and full-text search, respectively. Finally, Section 6 presents the conclusion and future work.

2 Related Work

The two kinds of researches related to our work are researches on content consistency maintenance and those on information search in a P2P environment.

In this paper, we refer to a hybrid P2P system, such as Napster that uses a central server, as a P2P system, although it is not entirely decentralized. This is because, even a hybrid P2P system has an important advantage in terms of content sharing; it can distribute large amounts of contents with less bandwidth consumption on the service providers side.

2.1 Contents Consistency Maintenance

Since the contents are stored on clients in a P2P content sharing system, malicious clients can tamper with the contents if no protection method against tampering is provided.

The MD5 hash function in the protocol of Napster [4] enables a content publisher to send the hash value of a content to a central server when it publishes the content. Freenet [2] prevents tampering with the content by using the hash value of a content as its key.

This technique is effective in preventing the tampering of static content such as a movie or music content. However, when this technique is applied to frequently updated contents, each version is treated as a separate content because different versions have different keys. To handle such frequently updated contents, Freenet introduced indirect files in which the hash values of the contents are stored. By retrieving an indirect file, a user can retrieve the last updated content in two steps. In order to share frequently updated contents, we need to provide a mechanism that associates the content ID with the hash value of a particular version of the content, as in the case of Freenet.

Another problem of P2P content sharing systems is that the provider of a content sharing service cannot trace the exchange of contents among users.
Napster, which is a centralized P2P content sharing system similar to our system, uses a download protocol by which the clients send a download request to the central server before they download the content from another client. After this, the central server does not participate in the download process of the content. Using this protocol, the central server cannot identify whether a download has been carried out successfully or not. A malicious client can send the same information to the central server and pretend that a download request has been made by another client. It is also possible to send tampered content to another client without being detected by the central server.

2.2 Information Search in P2P Environment

The two types of search techniques that are widely used in P2P content sharing systems include using a central search server [4] and flooding of search requests [6].

The problems of using a central server, such as poor scalability of a central search server and vulnerability that arises from a single point of failure, are widely known. The flooding of search requests also has scalability problems. As the number of nodes in a network increases, more search requests are flooded that consume a major part of the bandwidth. In order to reduce search requests, many systems use flooding techniques that often limit the search range with heuristic methods. As a result, it cannot be assured that all existing contents in a network can be found in these systems.

In order to solve the problems associated with the above mentioned techniques, several search methods based on distributed hash tables (DHT) have been proposed [5][7]. These methods are scalable to a considerable extent. A characteristic of these methods is that exact match key search can be done with $O(\log n)$ or $O(n^a)$ hops.

Reynolds and Vahdat proposed a method for implementing full-text search by distributing the reverse index on a DHT. In this method, a key in a hash table corresponds to a particular keyword in a document, and a value in a hash table corresponds to a document that contains a keyword. A client that publishes a document notifies the nodes that correspond to the keywords contained in the document and updates the reverse indexes on these nodes. In this method, the load of the full-text search can be distributed among the nodes. We can also expect that the reverse indexes on the nodes can be updated rapidly by pushing the latest keywords in the contents from a client.

On the other hand, this method has several limitations. For example, when an AND search is performed by this method, the search results must be transferred between the nodes. Li estimated the amount of resources that is necessary to implement a full-text search engine based on this method and pointed out that it is difficult to implement a large-scale search engine, such as Google, by this method [8].

Furthermore, if this method were applied to a P2P content sharing system, the problem of low availability of nodes would arise because the users' PCs would be used as nodes in such a system. In order to store reverse indexes on the nodes, we have to replicate them to ensure the availability of indexes. This would require more resources than that estimated by Li.

Based on the above mentioned reasons, we believe that a full-text search technique using a central search server that manages reverse indexes is more feasible than a distributed reverse index technique for implementing a full-text search engine in a P2P environment.

3 System Architecture

Figure 1 shows the architecture of our system. As described earlier, we chose a central server architecture to provide a full-text search of the contents.

The public keys of the clients are stored on a central server. By sending a request to the server, a client can obtain a public key of another client that is connected with the central server. The central server also has private and public keys. Its public key is available to all the clients.

Each client has a unique ID. When a client connects to the central server, it sends its own IP address. Another client can obtain the IP address of a client by querying to the server using its client ID. The central server provides a content consistency maintenance mechanism and a full-text search engine. These mechanisms are described in the following sections.

4 Content Consistency Maintenance

4.1 Data Structure for Content Management

In this system, a publisher of a document digitally signs a document with its private key and registers its sign to the central search server with its unique ID. When a document is a text document, a client performs morphological analysis to generate search keywords from a document.

The ID of contents and digital signs corresponding to different versions are managed on the central search server. Using the ID and version, a client can obtain a digital sign for a document by querying to the central server using its ID and version. Using a digital sign ensures that a malicious client does not tamper with a document.

A search result obtained from the central server is also digitally signed to ensure that a client does not tamper with it. As described in detail in section 5, a search result is cached on a client and can be modified. To prevent this, a search result comprises the ID of contents and a digital sign.

In this system, a client can obtain the latest version of a document when a document is updated, by querying its ID to the central server. However, a limitation associated with this method is that only the latest version of documents can be obtained. For example, by using indirect files and hash values of contents as in Freenet, we can obtain previous versions of a document by directly specifying a hash value of an earlier version. However, neither does Freenet assure that the latest version is always obtained nor does it assure that a particular earlier version is obtained because a previous version may be deleted if there is no request for it in a certain period. In our system, we consider only the latest version of a document which can be obtained at any time. Thus, we define our document query protocol in order to obtain the latest version.

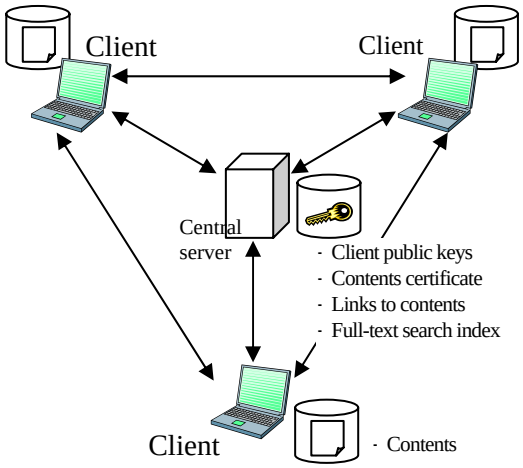

Figure 1. System Architecture

In order to prevent the concentration of download requests on a certain client, our system manages a list of clients that have downloaded the latest version of a document and distributes download sources to these clients using this list.

In this method, the ID of a client that downloads the latest version of a document is added to a list; this ID corresponds with the ID of the document. When a client sends a request to the central server to download a document, the central server selects an appropriate client from a downloader's list and returns its ID to the client. When the publisher updates a document, the list corresponding to that document is emptied.

We describe this procedure by the following pseudo codes, where `download` is a function that requests the download of a document, `nodeId` is the ID of a client that requests the download, `update` is a function that requests the update of a document, and `getNodeId` is a function that gets the ID of a client that downloads a document whose ID is `docId`.

`nodeIdList`: document ID x node ID list

```
download(docId, nodeId) {
  nodeIdList[docId].add(nodeId); }

update(docId, nodeId) {
  nodeIdList[docId] = {nodeId}; }

getNodeId(docId) {
  index = rand() * nodeIdList[docId].length;
  return nodeIdList[docId][index]; }
```

4.2 Tracing How Contents are Exchanged

In a P2P content sharing system that uses a simple download protocol, such as Napster, when a service

provider keeps records about how contents are transferred among clients, there exist possibilities of a client tampering with such records by sending false information about downloading content to the central server.

For example, by Napster protocol, a request to start a download that is sent when a client begins to download from another client is the information that the central server receives from the client. Therefore, the central server can obtain the same information in the case when a download source does not transfer a document as well as in the case when a download source transfers a document successfully.

To avoid this problem, our system uses a download protocol that employs the public keys of clients managed on the central server. The protocol is described as follows wherein a download destination client is denoted as client A and a download source client as client B.

1 Client A sends a download request to the central server. The central server generates a common encryption key and sends it to client B.
2 Client B encrypts the requested content with a common encryption key, signs it digitally, and sends the encrypted content to client A.
3 Client A confirms that the downloaded content has been signed by client B. Client A then sends a request for the common encryption key to the central server. The central server records that the content is downloaded.
4 Client A decrypts the downloaded content with the common encryption key. Client A then verifies that the downloaded content is not altered using digital sign on the central server.
5 If the downloaded content is altered, client A sends the downloaded data with a sign of client B to the central server. The central server can then confirm that it is signed by client B and is altered. The central server can then cancel the download record created in step 3.

By this protocol, the following properties are satisfied:

- A content download is recorded on the central server as long as a download source client follows the above protocol.
- The central server does not create a record when a document is not downloaded by a client.

When a download source client encrypts a document with a common encryption key following the protocol, a download destination client has to send a request for a common key to the central server. Thus, the central server can record a download. As a result, the first property is satisfied.

Further, when a client that downloaded a document sends a request for a common key, it obtains a sign of a download source client for the document. When the downloaded content is altered or different from the requested one, a download record can be cancelled by sending the downloaded data to the central server. Thus, the second property is satisfied.

However, even with this protocol, in the case when both a download source client and a destination client do not follow this protocol, the central server cannot record downloads of contents, for example, in the case where a download source client does not encrypt the content with a common key. Currently, our system does not handle such situations. We would like to consider this problem in our future work. In order to handle such situations, we evaluated the credibility of clients from download histories and selected a credible client as a download source.

5 Full-text Search

5.1 Load Balancing of Full-text Search

To reduce the load of full-text search on the central server, our system uses a caching technique to cache the search results of clients. It has been reported that approximately 30% to 40% of search requests are repeated on a full-text search engine [9]. Therefore, a caching technique is expected to considerably reduce the full-text search load. We employed an applied form of a hash-based caching method, as described in [3]. In this section, we describe the manner in which a full-text search is performed and search results are cached.

The central server selects a fixed number of clients as caches for the search results that connect for a long duration. The central server assigns them to the equally divided range of a hash function. A client obtains a list of caches when it connects to the central server. When a client performs a search, it calculates the hash value of a search keyword and sends a request to the cache assigned to a section of the range containing the hash value of the keyword.

In an experiment described later, SHA1 is used as a hash function for clients IDs and search keywords. By comparing several upper bits of hash values, we implement equally divided range of a hash function.

If a cache does not have a search result for a search keyword, it forwards the search request to the central server. The central server then returns the result to the cache with a search keyword, search time, and digital sign. The search time and digital sign that a client receives along with the search result from a cache can confirm that the result is not stale and not tampered with by a cache.

When a client sends a search request to a cache and detects that a cache is not available because it is connected to the network or is overloaded, the request sending client marks that it is not available on a list of caches. It then sends the search request to a cache assigned to the next section of the hash range. When the number of unavailable caches exceeds a certain

	max search requests	cache hit ratio	av. cache capacity
(1) 256	557	20.6%	75.7
(2) 1024	1193	24.3%	61.4
(3) 256×4	578	10.6%	42.1

Table 1: Result of Experiment

fixed number, it requests the central server for the most recent list of caches and updates it.

5.2 A Load Balancing Experiment

It is reported that the number of search requests for each keyword follows Zipf distribution [9]. Therefore, when we increase the number of caches by sub-dividing the range of a hash function, it is possible that search requests are concentrated to a certain cache.

On the other hand, if we increase the number of caches by maintaining the number of sections of the range of a hash function and increasing the number of clients assigned to each section, the cache hit ratio would decrease.

In this research, a preliminary experiment is performed in order to verify the advantages and disadvantages of these alternatives.

First, we generate a list of search keywords so that the number of requests for each keyword follows Zipf distribution. In this list, 40% of the queries are repeated and requests of the most frequently repeated word correspond to 0.2% of the entire range of requests. These numbers follow the search trace of Excite, as reported in [9].

If all search results corresponding to keywords in this list can be cached, the cache hit ratio would be 40%.

Using this list, we measure the cache hit ratio, required cache capacity, and the number of search request counts for each cache client in the following three cases: (1) the range of a hash function is divided into 256 sections, a single client is assigned to each section, and 100,000 requests are sent; (2) the range of a hash function is divided into 1024 sections, a single client is assigned to each section, and 400,000 requests are sent; (3) the range of a hash function is divided into 256 sections, 4 clients are assigned to each section, and 400,000 requests are sent. We assume that the same capacity is required to cache a search result for any keyword. Therefore, the number of keywords that are requested more than twice are counted as the required cache capacity.

Figure 2 shows the experimental results. This graph shows the number of requests to each cache that is sorted in a descending order. In order to compare these three cases, the scale of the x-axis of the result of case (1) is expanded by four times.

In case (1), although the range of the hash function is divided equally, the number of requests to the most frequently requested cache is less than twice that of

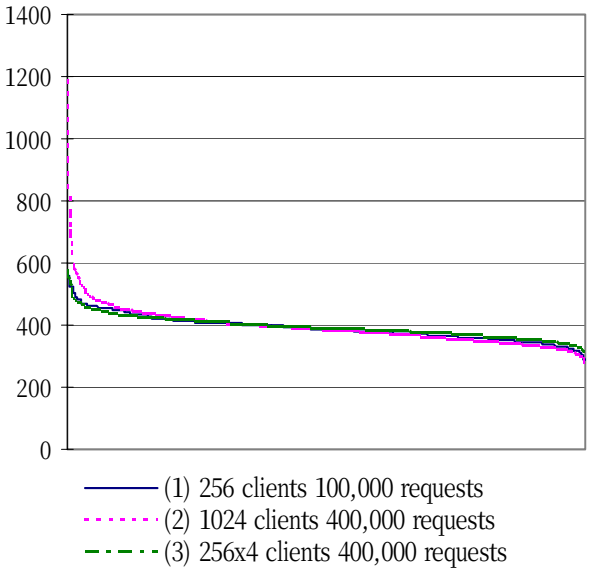

Figure 2. Distribution of Search Requests

the least frequently requested one. This shows that both the hash value of a frequently searched keyword and a not so frequently searched one are likely to be contained in one interval when the range is divided into relatively large intervals. Thus, the search requests for each interval are balanced. This result also shows that the cache hit ratio in case (1) is relatively high.

When the number of caches and search requests are increased from that in case (1) and when the range is divided into smaller intervals, as in case (2), search requests are concentrated to a certain cache, as shown in Table 1. This shows that the search load becomes unbalanced among the caches. However, in comparison to case (1), the cache hit ratio is improved because the number of search requests is increased in this case. As shown in Figure 1, the search loads are well balanced between most of the caches except a few caches where the search requests are concentrated in case (2). In order to balance the search load, we employed certain techniques to forward the search requests from overloaded caches to others. Currently, the above results only measure the number of search requests and do not consider the search load on caches. As a future work, we intend to perform a quantitative experiment of search load balancing under the condition when our system would forward search requests from overloaded caches to others.

When the number of clients assigned to each section increases, as in case (3), the search load is well balanced as shown in Figure 2, which is similar to that of case (1). However, the cache hit ratio decreases remarkably. In order to increase the cache hit ratio in this situation, other techniques such as hierarchical cache would have to be used.

In this experiment, we assume that the appearance ratio of repeated queries is the same. However, due to

limited vocabulary of the users, the ratio of repeated queries in the entire range of queries is expected to increase with the number of queries. Owing to this, we can expect that the cache hit ratio does not decrease when the number of queries increases.

6 Conclusion

In this paper, some problems regarding currently available P2P content sharing systems such as content consistency maintenance and information search have been pointed out. We proposed techniques in order to solve these problems and described the outline of the P2P content sharing system that we are developing.

This system uses a central server on which the digital signs of contents publishers are maintained to prevent the tampering of contents. Further, this system adopts a content transfer protocol that employs asymmetric encryption keys. This protocol enables us to record content exchanges between clients on the central server. On the contrary, since most other P2P content sharing systems do not employ sufficient techniques to maintain records of content exchanges, such records would be unreliable. In particular, it is difficult to maintain such records in decentralized P2P content sharing systems.

In order to solve the problem regarding information search, we propose full-text search techniques for P2P content sharing systems. First, morphological analysis and summarizing of documents are performed for each client and the results are sent to the central server to generate reverse indexes. We can implement a relatively efficient full-text search with this technique. More efficient and scalable full-text search techniques based on DHT are currently being researched. It is difficult to implement partial matching or AND search by these techniques; however, they can be easily implemented with our method. In order to reduce the search load on the central search server, we propose a load balance technique that caches search results on clients and uses a hash function to distribute search requests to clients. In one of the experiments carried out, it was seen that the search requests were satisfactorily balanced, and the cache hit ratio was relatively high for a considerably large set of search requests that follow Zipf distribution.

We believe that content consistency maintenance and efficient full-text search on P2P content sharing systems can be implemented using our techniques.

As a future work, we would like to consider novel techniques to handle cases where multiple clients do not follow our content transfer protocol. In addition, we believe it is necessary to quantitatively evaluate our implementation to confirm that our system functions well in a practically large-scale environment. With respect to the caching technique, we have to improve our load balancing technique to avoid some clients from being overloaded.

Acknowledgements

Hideki Mima and Hideto Tomabechi express their gratitude to the Ministry of Internal Affairs and Communications for promoting the study in part under the SCOPE R&D grant scheme.

References

[1] P. Reynolds and A. Vahdat. Efficient Peer-to-Peer Keyword Searching. *Middleware 2003*.

[2] T. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *ICSI Workshop on Design Issues in Anonymity and Unobservability*, 2000.

[3] David R. Karger, Eric Lehman, Frank Thomson Leighton, Rina Panigrahy, Matthew S. Levine, and Daniel Lewin. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web. In *ACM Symposium on Theory of Computing*, pages 654-663, 1997.

[4] Napster. http://www.napster.com/, http://opennap.sourceforge.net/.

[5] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. In *Proceedings of ACM SIGCOMM'01*, 2001.

[6] Gnutella. http://gnutella.wego.com/.

[7] Tylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content-addressable network. In *Proceedings of ACM SIGCOMM'01*, 2001.

[8] Jinyang Li, Boon Thau Loo, Joseph M. Hellerstein, M. Frans Kaashoek, David R. Karger, Robert Morris. On the feasibility of peer-to-peer web indexing and search. In *2nd International Workshop on Peer-to-Peer Systems*, 2003.

[9] Yinglian Xie and David O'Hallaron. Locality in search engine queries and its implications for caching. *IEEE Infocom 2002*, 2002.

[10] Yasuaki Takebe, Hideki Mima, Hideto Tomabechi. A next-generation P2P contents sharing system–implementing content consistency maintenance and full-text search. In *11th DPS Workshop*, 2003.