Mimir: a term-distributed retrieval system for secret documents

Guoqiang Gao*
School of Media and Communication,
Wuhan Textile University,
Wuhan, Hubei, China
Email: ggao@wtu.edu.cn
*Corresponding author

Ruixuan Li
School of Computer Science and Technology,
Huazhong University of Science and Technology,
Wuhan, Hubei, China
Email: rxli@hust.edu.cn

Zhiyong Xu
Math and Computer Science Department,
Suffolk University,
Boston, Massachusetts, USA
Email: zxu@mcs.suffolk.edu

Abstract: In order to access sensitive documents shared over government, army and enterprise intranets, users rely on an indexing facility where they can quickly locate relevant documents they are allowed to access: 1) without leaking information about the remaining documents; 2) with a balanced load on the index servers. To address this problem, we propose Mimir, a distributed cipher retrieval system for sensitive documents. Mimir constructs the distributed indexes based on load balanced term distribution for better search efficiency and load balanced query. Mimir utilises encryption with random key, partial key update to protect sensitive data and improve query efficiency. Our experiments show that Mimir can effectively protect secret data and answer queries nearly as fast as an ordinary inverted index.

Keywords: ciphertext retrieval system; index; search; term distribution; encryption.


Biographical notes: Guoqiang Gao received his MS and PhD in Computer Science from Huazhong University of Science and Technology, China in 2007 and 2011, respectively. He is currently an Associate Professor in School of Media and Communication at Wuhan Textile University. His research interests include peer-to-peer computing, cloud computing, and data mining. He is a member of IEEE.
1 Introduction

The number of secret documents shared over government, army and enterprise intranets is growing rapidly. How to effectively manage these secret documents? Building an inverted index over the collection of secret documents may be a good choice (Zhao et al., 2014). The inverted index consists of a couple of data structures, namely lexicon and posting lists. The former stores all the distinct terms who appear in the documents, and the latter is an array of list storing terms occurrences within the document collection. Current most search engines use the inverted index to keep information, such as Google. However, storing plain text in the inverted index is not secure for secret documents because an adversary who gains access to the index files can access sensitive data, thus, bypassing the access control mechanism. Encrypting the index can prevent data from being illegally stolen, while it will lead to large computational overhead. In order to obtain high efficiency, we can only encrypt the index’s lexicon because the document can be only derived from it. Although the term appears in the index as ciphertext, there is still a risk of statistical attack for this strategy because the highly frequent terms have a long posting list. To put it simply, the attackers can learn the rule of posting list from the frequency of terms. Then they can estimate the encrypted terms through the length of the posting lists. In this paper, we propose a strategy that a term is encrypted with a random key. Thus, even if the key of a term is cracked, it will not pose a threat to the other terms.

The amount of sensitive documents stored in some secret intranets reaches at the terabyte range. If their data is stored and indexed on a single computer, the queries will take many seconds to evaluate even with the most efficient index representations and query resolution methods. To handle the necessary data volumes and query throughput rates, parallel computing systems are used, in which the documents and index data are split across tightly-clustered distributed computing systems. The main distributed indexing methods include document-distributed index (Kulkarni and Callan, 2010) and term-distributed index (Cambazoglu et al., 2013). Each processing server stores the index corresponding to a subset of the documents in a document-distributed system. Queries are
Mimir: a term-distributed retrieval system for secret documents

processed in parallel at all servers, and collated back into a single combined answer when all servers have completed their local processing. The document distribution has good scalability and load balancing, while it has low search efficiency. In a term-distributed system, each of the processing servers maintains complete index information for a subset of the terms in the collection, and each query is referred to the subset of the servers that hold relevant information. The term distribution has compact index and high search efficiency, while there is an evident lack of balance in the distribution on the load of its servers (Badue et al., 2007). In this paper, we propose a load balanced term distribution strategy to retain the advantages of term distribution, and to improve the load unbalance.

A distributed retrieval system usually consists of the following components: clients, the receptionist and query processors. A receptionist, is called as the management server in our system, receives queries from client computers, and makes decisions on how to route these queries to different query processors. The query processors, or so-called index servers, hold index or document information, which are used to retrieve and prepare the presentation of results, respectively. In a standard query resolution method, the receptionist receives a query, requests the index information for the query terms from all corresponding index servers, receives query results and merges them, and return the final result to the client. To address the problems discussed above, we develop Mimir, a distributed retrieval system for secret documents. We use a P2P-based distributed system to store massive amounts of sensitive documents. In order to improve security, Mimir uses the cipher index to protect sensitive information, and encrypts the terms in the index with random key to reduce the risk of statistical attack, and takes partial key update strategy to decrease the potential risk of attack by malicious users. To improve system performance, Mimir only encrypts the terms in the index to decrease the cost of decryption and improve the retrieval efficiency, and constructs the distributed index based on the load balanced term distribution for secret documents to balance the load of the index servers. Mimir not only guarantees the safety of secret documents, but also the efficiency of the retrieval system.

This paper is organised as follows: in Section 2, we describe Mimir in details from load balanced term distribution, random key and partial key update. Section 3 evaluates the performance through experiments. Related work is presented in Section 4. We conclude and summarise the results in Section 5.

2 Mimir distributed cipher retrieval system

Mimir proposed by us is a distributed retrieval system for secret documents, that has two main functions: distributed indexing and distributed searching. We will discuss each feature of Mimir in details after an overview of its architecture.

The components of Mimir include the clients, the management server, the index servers, the random key database (RKDB) and the encryption server as shown in Figure 1. In the client (web browser), an authorised user can log onto the management server to submit a query, and the administrator can login the management through the client as well and build an index. The management server distributes the entire index to p index servers as well as r secret documents to p index servers, and assigns a query to the corresponding index servers to perform search. To improve search efficiency, Mimir uses the term distribution strategy to split the entire index. In indexing, a document is parsed into a series of terms that are then assigned to the corresponding index servers. In the
index servers, Mimir uses the cipher terms, which are generated by encrypting the plain terms, to build the index, and this index is called the cipher index. All the cipher indexes stored in the index servers constitute the distributed cipher index that is complete logically. The index server is also responsible for handling queries, so it also can be called the query server. When a query is launched, it will be parsed into multiple query terms, and then these terms are sent to the corresponding index servers to perform queries, and finally the results are transferred back to the management server to merge. The index servers are also responsible for keeping the documents used to build the distributed index. The plain documents are assigned to corresponding index server to keep according to some rules after they are encrypted. In the indexing and querying processes, the terms are encrypted through the encryption server, and a term obtains the key stored in RKDB through using its hash value. In the front end, Mimir uses other application with access control policy to protect sensitive data from unauthorised access; in the back end, encryption takes care of data leakage from illegal use. In the following, we will introduce the key features of Mimir in details.

Figure 1  The architecture of Mimir (see online version for colours)

2.1 Load balanced term distribution

We will first discuss the index distributing strategy, where Mimir uses a load balanced term distributed mechanism to distribute the index. Although the term distribution has better search efficiency, such as less disk access, than the document distribution, there is an evident lack of balance in the load distribution of the index servers. Suppose that $r$ documents are parsed into $m$ terms, then the number of terms $s_i$ that are assigned into an index server in term distribution, is refined in equation (1).

$$s_i = \frac{m}{p}$$  \hfill (1)  

where $p$ is the number of index servers. Therefore, each index server is assigned the same number of terms. Suppose the index server $I_a$ owns the terms set $T_a = (t_1, t_2, \ldots, t_n)$, and the frequency of a term $t$ is defined as $f(t)$. The term frequency of index server $I_a$ is refined in equation (2).
Since the frequencies of terms are different, according to Zipf’s law (Bochkarev and Lerner, 2012), the term frequency of each index server is also different even with the same number of terms. A higher frequency term results in more documents containing it whose posting list may be a bit longer. If \( f(I_4) > f(I_6) \), the index size of server \( I_4 \) is bigger than the size of server \( I_6 \). In the query, the load of index server \( I_4 \) will be greater than that of \( I_6 \).

In this paper, to ensure load balancing, the load balanced term distribution will be adopted instead of random term distribution. Most of the terms are low frequency terms, and imbalanced load caused by the low-frequency terms is small because the frequency interval between them is very little. In order to improve efficiency, the low frequency terms are assigned to the index servers with a random term distribution strategy. For the high-frequency term, the management server records the \( f(I) \) of each index server, where \( f(I) \) denotes the high-frequency term frequency of the index server, and the management server establishes a list \( L(\text{term, index server}) \) for the allocated terms. In the distribution process, the low-frequency words were randomly assigned to the index server. If a high-frequency term \( t \) is not in the list \( L \), it would be assigned to an index server \( I \) with minimal \( f(I) \), and the management updates \( f(I) \) by \( f(I) = f(t) \). The details are presented in Algorithm 1.

**Algorithm 1**  
Load balanced term distribution (term \( t \))

```
if \( f(t) > f_{\text{threshold}} \) then
    if \( t \) not in \( L \) then
        \( I = \{ I \mid \min(f(I)) \} \)
        allocate \( t \) to index server \( I \)
        \( f(I) = f(t) \)
        insert \( t \) and \( I \) into \( L \)
    else
        lookup \( t \) in \( L \) and get \( I \);
        allocate \( t \) to index server \( I \)
    end if
else
    \( I = \text{hash}(t) \mod p \)
    allocate \( t \) to index server \( I \)
end if
```

### 2.2 Random key mechanism

The terms are the most important part of the inverted index because the documents can be deduced from them. To ensure security of document data, it is necessary to ensure the safety of inverted index. The best way is to encrypt the index as a whole, but it would greatly reduce the efficiency of retrieval. In this paper, we only encrypt the terms in the index in order to make a system both safe and efficient. To process the massive collection
of secret documents, we can build cipher index over multiple index servers. The
distributed cipher index proposed in this paper is described in Figure 2. The cipher index
which is wholly logical is separated into multiple index servers through term distribution.

Figure 2  The structure of the distributed cipher index with six terms

As previous discussion, there is a high risk of statistical attack because the posting lists
are not encrypted. The high frequency terms have longer posting lists, so the plain term
can be estimated from the ciphertext term through the length of its posting list. For
example, according to the British National Corpus (BNC), the word ‘people’ has the
highest frequency as an available indexing term. Therefore, the cipher term with the
longest posting list can be estimated as the word ‘people’, and the encryption key can be
obtained from the cipher term and the plain term. Then the other terms in the cipher index
can be decrypted through the cracked key. To solve this problem, we propose an
approach that encrypts or decrypts terms not with the same key but with random key. A
term obtains an encryption key from RKDB based on its hash value. In this case, even if
a term is cracked, the others will not be affected because the terms have different
encryption key. Therefore we can significantly reduce the risk of statistical attack through
using this strategy.

In Mimir, RKDB has two functions: to store the key and protect the key, as a result,
RKDB must be an encrypted database. Table 1 shows the data structure storing the key.
Serial number (SN) indicates the key position in RKDB, and a term can obtain its key
according to SN. The ‘Temp Key’ field temporarily stores the new key during key
update, and replaces ‘Key’ with it when the index update is complete. The ‘Terms’ field
in Table 1 indicates the terms set using the corresponding key. This information is used
for partial key update which will be discussed in the following section. Supposing the
number of key stored in RKDB is M, then the method that a term obtains a key is: the
hash value of the term mod M and obtain the key from RKDB according to the result. For
example, the hash value of ‘Chinese’ is 1,031, and M is 5,000, thus the key for ‘Chinese’
can obtain from RKDB through the remainder 1,031. At the same time, the word
‘Chinese’ will be insert into the corresponding terms set if the set does not contain it. The
method of obtaining a key from RKDB is presented in Algorithm 2.
Table 1  The data structure for key

<table>
<thead>
<tr>
<th>SN</th>
<th>Key</th>
<th>Temp key</th>
<th>Terms</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>key0</td>
<td></td>
<td>computer;...</td>
</tr>
<tr>
<td>1</td>
<td>key1</td>
<td></td>
<td>cipher; index;...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Algorithm 2  Obtain key (term t)

\[
\begin{align*}
  sn &= hash(t) \mod M \\
  key &= RKDB[sn].Key \\
  \text{if } t \text{ not in } RKDB[sn].Terms \text{ then} \\
  \quad \text{add } t \text{ to } RKDB[sn].Terms \\
  \text{end if} \\
  \text{return } key
\end{align*}
\]

2.3 Partial key update and secondary index

Although random key reduces the risk of statistics attack, the correspondence between term and key is fixed. That is the key of a term is fixed unless the key is changed. To reduce the potential risk, the key must be frequently updated. For Mimir, when the key of a term is updated, the corresponding cipher term will be changed. This will lead to huge system overhead because of reconstruction of the term table in cipher index. We only update partial key in RKDB randomly each time, which is called partial key update by us. Although the key is not updated fully, random updating can disrupt the correspondence between term and key, thereby, enhancing the security of the system. In Mimir, each time we only update 10% key in RKDB by random to improve updating efficiency and ensure system security.

In order to improve the efficiency of key updating, Mimir do not reconstruct the entire cipher index. If a key is updated, firstly, Mimir finds the corresponding terms of it through RKDB. Secondly, Mimir obtains the index servers of these terms through Algorithm 1, and encrypts these terms using old and new key to gain old and new cipher terms. Finally, Mimir sends these cipher terms to the corresponding index servers to update index. In index servers, Mimir cannot simply replace the old cipher term with a new cipher term, otherwise the order of the term table in index will be upset. Mimir locates the terms position through the old cipher terms and marks the terms as deleted terms firstly. Then, Mimir builds a secondary cipher index for the terms whose key have changed. Since the posting list has not changed, the index building only is to sort the term table. Suppose to take bubble sort, the comparison times of terms for entire index rebuild is refined in equation (3).

\[
\sum_{i=2}^{n} (i - 1) = n \frac{(n - 1)}{2}
\]

where \( n \) is the number of terms in index. Therefore, the time complexity of entire index rebuilding is \( O(n^2) \). As Mimir only updates 10% key each time, that is nearly the cipher text of 10% terms are changed, Mimir has only one tenth of the overhead compared to
entire index rebuild. However, there is the cost of index merge because Mimir generates the secondary index. As the posting list does not change, this cost only comes from the merging sort of two terms table in fact. The time complexity of index merge for key update is $O(1)$. To improve system performance, for key update, Mimir initiates the operation of index merge at idle only when the number of the secondary indexes reaches a certain threshold.

2.4 Using Mimir

In this section, we will describe two main functions in Mimir: distributed indexing and searching.

2.4.1 Distributed indexing

To index a document, the management server extracts the document’s terms, obtains their key with Algorithm 2 and encrypts them in the encryption server, gets their index servers with Algorithm 1 and sends the cipher terms to the corresponding index servers to build index. The document is encrypted as a whole, and then is stored in a index server according to the hash value of its title name. Meanwhile, the management server extracts the access control bitmap of this document for role and user from the access control index after it is decrypted, which is used to verify the uses and filter query results.

In order to update a key, the management server finds the position of the key from RKDB, inserts the new key into the corresponding position of ‘Temp Key’, and reads the terms who correspond to the key firstly. Secondly, Mimir determines the index servers of the terms according to Algorithm 1, encrypts the terms with old key and new key, and sends the cipher terms into the corresponding index server to update index. Thirdly, in the index server, Mimir locates the terms position through the old cipher terms, marks the terms as deleted terms in the index, and builds a secondary cipher index for the terms whose key are changed. Finally, the index servers return the result, the index update is finished, to the management server, and Mimir replaces the key with the new key stored in the ‘Temp Key’ filed. That the number of secondary index reaches a certain threshold will trigger Mimir to merge indexes at idle.

2.4.2 Distributed searching

Firstly, the management server authenticates the user when a user lunches a keyword query. If the authentication is passed, Mimir parses the query into multiple terms, and the parsing method is the same with indexing. To assign the retrieval terms to correct index servers, the distribution strategy which gets the index server location of the term based on the term hash value is similar to indexing. The method that a retrieval term obtain its corresponding index server is described in Algorithm 3. Then the retrieval terms obtain their key through Algorithm 2, and are encrypted into the cipher retrieval terms at the encryption server. Secondly, the management server sends the cipher retrieval terms to the corresponding index servers to perform searching. In the index servers, also can be called query servers as previous description, the cipher retrieval terms are searched over cipher index, and the result is returned to the management server. Finally, the management server receives the search results from the multiple index servers, which will be merged, and shows the final results to the users.
If the users want to view or download their interested document, the management server will get this document from the corresponding index server and send it to the users. Compared to the document distribution, Mimir performs distributed search only over part of the index servers. Therefore, Mimir reduces the load on the system effectively. Since only the terms are encrypted, the search strategy of Mimir is the same with the plain retrieval engines, where Mimir encrypts the query terms, and search these cipher terms from the ciphertext index. Compared to the overall encrypted index, Mimir greatly improves the retrieval efficiency. Meanwhile, the encryption and the access control can guarantee the security of the secret documents as well.

**Algorithm 3** Obtain index server (term \( t \))

```
if \( f(t) > f_{\text{threshold}} \) then
    lookup \( t \) in \( L \) and get \( I \);
else
    \( I = \text{hash}(t) \mod p \)
end if
return \( I \)
```

3 Experiment setup

In this section, we first discuss the experimental hardware and software settings, and then introduce the collection used in our experiments.

3.1 Hardware and software

The hardware used in the experiments is seven computers distributed on our Laboratory LAN. Each computer has a 2.5 GHz Intel Xeon with 2GB RAM and 1TB local SATA disk in a RAID-5 configuration. One of the computers acts as the management server in which the encryption card replacing the encryption server and RKDB are installed, and the other six computers are acted as the index servers. These computers are interconnected through 100 Mbps Ethernet. We install Redhat Linux AS 4 on each machine as the operating system, Tomcat 6.2 as the web server, Java 1.6 as development and runtime environment.

The distributed cipher retrieval system Mimir has been developed at Huazhong University of Science and Technology using above ideas discussed in Section 3. Mimir was developed using java, including Mimir master and Mimir slave. Mimir master is a Browser/Server program, and installed at the management server. Mimir master can be deployed to a variety of web servers. In our experiment, we take tomcat as the deployment platform. Mimir slave is installed at the index servers. When Mimir initialisation, Mimir slave will start the indexing and query services. Mimir master communicates with Mimir slave by calling these services. For the indexing and query requests, Mimir master firstly divides these requests into multiple sub-requests based on distribution strategies discussed above. And then Mimir master assigns these sub-requests to different Mimir slaves to handle by calling the indexing and query service. Finally, Mimir master merges the results returned by Mimir slaves, and shows them to the users. Figure 3 shows the software architecture of Mimir.
3.2 Collection and queries

In our preliminary research (Gao et al., 2010), we only use an English collection to evaluate Mimir performance. In this extended study, for a more comprehensive analysis, we used two collections for our experiments, which we refer to as Reuters Corpus Volume 1 (Reuters-RCV1) and Sougo news data (SougoCS). Reuters-RCV1 consists of about 800,000 documents that were sent over the Reuters newswire during a one-year period between August 20, 1996, and August 19, 1997. Reuters-RCV1 covers a wide range of international topics, including politics, business, sports, and science. Many studies have used Reuters-RCV1 as the test collection. In order to evaluate the performance of Mimir on Chinese support, we used a Chinese collection: SougoCS, which contains 1,800,000 news pages from the Olympics, sports, IT, domestic, international and other 12 channels of SOHU, one of China’s major websites, on January to June 2008. For both collections, HTML and punctuation was removed from the content before indexing to minimise the impact of parsing on performance. Table 2 summarises the two collections used, and lists some relevant statistics for them.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Collection</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Reuters-RCV1</td>
</tr>
<tr>
<td>Size (GB)</td>
<td>2.5G</td>
</tr>
<tr>
<td>Documents (106)</td>
<td>0.81</td>
</tr>
<tr>
<td>Terms (106)</td>
<td>0.4</td>
</tr>
<tr>
<td>Cipher Index (GB)</td>
<td>2G</td>
</tr>
</tbody>
</table>
A experimental problem in information retrieval research is finding suitable query sets. In order to more effectively achieve our experiments, the query words do not come from a real-world query log. We randomly extract terms from the term set of the collections to compose the query words for simplicity. To assess the query performance, we use the one-term query, two-term query, and three-term query to search Mimir, and then calculate the average. For each type of query, we repeat 100 times with different query words to ensure the accuracy.

4 Evaluation

In this section, we discuss Mimir’s security guarantees and then discuss its load compared to the document distribution and normal term distribution, indexing performance and query performance.

4.1 Security guarantees

The encryption scheme can protect data privacy and data authenticity against adversaries that have access to the index on a low enough level to bypass the access control scheme. In Mimir, the secret documents are encrypted and kept in a group of index servers based on distributed networks, can also be called intranets. The terms, as cipher text way, appear in the distributed index for the secret documents and the posting lists are not encrypted to improve search efficiency. As the posting list does not contain the document content, and the risk brought by plain posting lists can be reduced through taking random key strategy. Therefore, the encryption strategies in Mimir can protect data against illegal access to the index. Unlike general retrieval engines, Mimir is used in the intranets, and furthermore, Mimir uses other application with access control policy to protect sensitive data from unauthorised access. As a result, Mimir can guarantee that the secret information will not leak through its platform.

The encrypted storage for the secret documents and the cipher index can guarantee the sensitive information, even the encrypted data is obtained by the attackers. As the posting list of the cipher index has not been encrypted, there is the risk of statistical attacks. Mimir reduces this threat through taking random key (RK) strategy. A term obtains its encryption key based on its hash value from RKDB, so the key for encrypting terms is different with each other. Even a key for a term is cracked according to the frequency of the term, it cannot be used to crack the other encrypted terms. RK strategy greatly increases the difficulty of attack. Hence, Mimir can guarantee data security, even that these data have been illegally stolen. Partial key update can disturb the correspondence between term and key, and is low cost because Mimir does not rebuild the whole index. Therefore, partial key update can be done anytime to enhance system security further.

4.2 Load balance

The major issue for throughput, in fact, is an uneven distribution of the load across the index servers. Figure 4 illustrates the average busy load for each of the six index servers of a document distributed system [Figure 4(a)], a term distributed system [Figure 4(b)] and Mimir [Figure 4(c)]. The dashed line in each of the three plots corresponds to the
average busy load on all the servers. For document distributed systems, the majority of the proposed approaches in the literature adopt a simple approach, where documents are randomly distributed, and each query uses all the index servers. Therefore, distributing documents randomly across index servers can guarantee an even load balance. However, the document distribution system has the largest communication overhead compared to the other two methods. In the case of the term distributed system, there is an evident lack of balance in the distribution on the load of the index servers, which has a negative impact on the system throughput. To overcome this issue, Mimir uses load balanced term distribution that would ensure the access frequency of the index in each index server is similar. The total load of Mimir is the same as the term distributed system, while Mimir has better system throughput. Mimir and document distributed system all have balanced load, but Mimir has less communication cost. Therefore, Mimir has a better load balance and efficiency compared to document distributed systems and term distributed systems.

Figure 4  Distribution of the average load per node in document distribution, term distribution and Mimir (see online version for colours)

4.3 Indexing performance

In order to discuss the efficiency of the index build, we analyse the indexing throughput which is the number of articles whose volume is about 2k to be indexed in one second, and the indexing time which is the total time spent on all tasks performed to index a collection of a given size. In order to comparison, the maximum index established by us only contains 800,000 articles (web page) because Reuters-RCV1 has fewer documents compared to SougouCS and it only has 810,000 documents. Figure 5 shows the throughput of indexing of Mimir with six index servers for collections Reuters-RCV1 and SougouCS. We found that Mimir has similar indexing throughput for both English and Chinese collections. The average indexing throughput of the Reuters-RCV1 is 161 documents/second and its performance is slightly better than that of SougouCS. We also found the indexing throughput of Mimir is independent of the index size. In a short, Mimir has a better performance of indexing regardless the type of the collections.

In order to further reveal the performance of Mimir, we take Mimir with plain indexing (PI) and block encryption indexing (BEI) to conduct a comparative analysis. PI is no encryption inverted index strategy which is commonly used. BEI divides the entire index into multiple blocks, and then encrypts each block as whole, which is another project of our research item. Figure 6 shows the throughput of indexing with three different strategies using Reuters-RCV1. The collection SougouCS shows similar results. Since Mimir only encrypts the critical terms, as a result, the encryption overhead
introduced by it is relatively small. The throughput of plain indexing can reach 180 documents per second and Mimir is closer to PI. As a large number of encryption overhead, the throughput of EPI is relatively low. Of course, EPI has higher encryption strength and can provide more security for confidential information.

Figure 5  The throughput of indexing of Mimir with Reuters-RCV1 and SougouCS (see online version for colours)

![Throughput of Indexing](image)

**Figure 6**  The throughput of indexing with different strategies (see online version for colours)

![Throughput of Indexing with Different Strategies](image)

Figure 7 shows the indexing time of dynamic index servers with three different strategies for the collection Reuters-RCV1. It is clear that as the number of dynamic index servers increases, indexing time decreases. Mimir needs about 85 minutes to construct the distributed index of the Reuter Corpus collection in six index servers, and the indexing time of it is close to PI. However, the indexing time of BEI is almost twice as much as Mimir for its overall encryption. Although Mimir does not encrypt the index as a whole, it also can ensure the security of the data through some mechanisms discussed above, Mimir can maintain the similar efficiency with plain indexing, and guarantee the data security as well.
4.4 Query response time

We evaluate effectiveness of query processing by analysing latency of queries in which query latency is represented by the average response time required to process a query. Figure 8(a) shows the query response time of Mimir with six index servers using the collections Reuters-RCV1 and SougouCS. We found Mimir has similar performance in query delay for both English and Chinese collections. The average query respond time of Mimir is about 0.33 second which can be acceptable by almost all persons. We use the queries with one term to ten terms to lunch the experiment, where the number of documents is 800,000. The experimental results are shown in Figure 8(b), and we can see that our system has better scalability, where the response time is located between 0.3 and 0.4 second for different collection and queries, which can meet our design goals.

In order to evaluate query performance for different information retrieval systems with different index servers, we use Reuters-RCV1 as the collection to perform the experiment for Mimir, PI and BEI. The collection SougouCS has similar performance as previous experiments. The results are shown in Table 3. We found the query performance decreased with the increase of the index servers. However, the good news is that this decline is not linear, and the range is not great as well. To further explain this phenomenon, we divide the query time into the transaction time and the search time, and use Figure 9 to describe the results for Mimir. The other applications show similar results. The transaction time includes the communication time and process time (preprocess, parse, distribute, encrypt and etc.). The search time indicates the time used to search index. As we can see from Figure 9, the search time with different index servers is almost the same. However, with the increase of the index servers, the transaction time increased gradually. This can explain the results in Table 3.
**Figure 8** The query response time of Mimir with Reuters-RCV1 and SougouCS (see online version for colours)

![Query response time of Mimir with Reuters-RCV1 and SougouCS](image)

**Figure 9** The transaction and search time of Mimir (see online version for colours)

![Transaction and search time of Mimir](image)
### Table 3  The query performance with different index servers

<table>
<thead>
<tr>
<th>Applications</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>BI</td>
<td>0.13s</td>
<td>0.15s</td>
<td>0.16s</td>
<td>0.17s</td>
<td>0.18s</td>
<td>0.19s</td>
</tr>
<tr>
<td>PEI</td>
<td>0.49s</td>
<td>0.53s</td>
<td>0.55s</td>
<td>0.56s</td>
<td>0.57s</td>
<td>0.58s</td>
</tr>
<tr>
<td>Mimir</td>
<td>0.26s</td>
<td>0.29</td>
<td>0.31s</td>
<td>0.31s</td>
<td>0.32s</td>
<td>0.33s</td>
</tr>
</tbody>
</table>

Figure 10 represents the query response time of Mimir, PI and BEI that use the collection Reuters-RCV1. The query response of Mimir and PI is not affected by the size of the index for the inherent retrieval efficiency of inverted index. The query response time of Mimir can be achieved on average 0.3 second and this can meet application requirement of cipher retrieval. Due to decrypting the corresponding block of search terms, the query response time of BEI will increase with the size of the collection.

**Figure 10** The query response time with different algorithms (see online version for colours)

5 Related work

Index security has been addressed by many researches, where the goal is to secure the secret documents from unauthorised access. Typically a system must provide two ways to achieve security: access control (Kim et al., 2014) and data encryption (Fathi et al., 2013). In this paper, we focus on the encryption that is a standard technique for storing data confidentially (Kallahalla et al., 2003). Bertino et al. (2001) provide a framework for policy-based protection of XML data by encryption. Seitz et al. (2003) proposed an architecture that allows users to store and share encrypted data in the grid computing environment. In order to improve the search efficiency of encrypted documents, the searchable encryption has been proposed. Traditional searchable encryption (Zhang et al., 2014; Liu et al., 2014; Bao et al., 2008) has been widely studied in the context of cryptography. Among those works, most are focused on efficiency improvements and security definition formalisations.
The first construction of searchable encryption was proposed by Song et al. (2000), in which each word in the document is encrypted independently under a special two-layered encryption construction. To achieve more efficient search, Chang and Mitzenmacher (2005) and Curtmola and Kamara (2006) both proposed similar index approaches, where a single encrypted hash table index is built for the entire file collection. In the index table, each entry consists of the trapdoor of a keyword and an encrypted set of file identifiers whose corresponding data files contain the keyword. As a complementary approach, Boneh et al. (2004) presented a public-key-based searchable encryption scheme, with an analogous scenario to that of Song et al. (2000).

Managing the documents with the inverted index can greatly improve search efficiency. Therefore, another idea of searchable encryption is to first construct inverted index for the sensitive documents, and then encrypt the inverted index. Each posting list element in Zerber (Zerr et al., 2008) is encrypted, while the terms in the index are not encrypted. This strategy will bring great decryption cost because the posting list accounts for a large proportion of the index, and may leak certain confidential terminology. In Mimir, we perform the opposite operation. In order to obtain the similar efficiency to an ordinary inverted index, the posting list is not be encrypted and Mimir only encrypts the terms in the index. In the absence of encryption posting lists, there is the risk of statistics attack in Mimir. To decrease this risk, Mimir takes random key to encrypt the terms. As a result, the terms have different encryption key to encrypt and then this strategy greatly enhanced the difficulty of crack. To further improve system security, Mimir disturbs the correspondence between term and key though partial key update. The variety of cipher terms caused by key change will result in index update. For index update, Margaritis and Anastasiadis (2009) only flush selectively the terms with most posting lists in memory into disk to merge it with primary index when the memory gets full with new posting lists. Gurajada and Kumar (2009) propose a new merge-based index maintenance strategy for information retrieval systems. This strategy partitions the index into frequent-term index and infrequent-term index based on the frequency of terms, and uses a lazy-merge strategy for maintaining infrequent-term index and an active merge strategy for maintaining frequent-term index. In our partial key update, Mimir builds secondary index for the terms whose key are changed, and initiates the operation of index merge at idle only when the number of the secondary indexes reaches a certain threshold.

To handle the large-scale secret documents, the secret documents and index should be distributed to multiple index servers. There is a substantial literature on distribution methods. Li et al. (2010) distribute index contains fuzzy keywords and encrypted files into the cloud servers. This strategy can provide an effective fuzzy keyword search over encrypted cloud data. According to the principle of confidentiality, however, the sensitive data should be stored in its own dedicated servers. The distribution methods can be broadly categorised into two types: document distributed and term distributed schemes. Harman et al. (1991) described a document distributed system that was successfully deployed in practice. Cahoon et al. (2000) found that increasing the number of nodes used to manage a fixed-size collection could improve response, with diminishing returns. Probably the best-known document distributed system is Google (Barroso et al., 2003), in which the cluster of servers maintains a document distributed index and other servers store information such as the documents themselves. The major drawback of document partitioned system is that servers execute operations unnecessarily when querying sub-collections, which may contain only few or no relevant documents. In term distribution researches, Baeza-Yates et al. (2007) represent a collection of documents
with a binary matrix \((D \times T)\), where rows represent documents and columns represent terms. Each element \((i, j)\) is ‘1’ if the document \(i\) contains term \(j\), and it is ‘0’ otherwise. The term partitioning consists of performing a vertical partitioning of the \(T \times D\) matrix. MacFarlane et al. (2000) found the overhead at the top process is a serious bottleneck with term-distributed mechanism. Since the major overhead comes from the search for retrieval systems, we can balance this problem through the pipelined search which will be discussed later. Moffat et al. (2007) showed that term partitioning resulted in lower utilisation of resources. More specifically, it significantly reduces the number of disk accesses and the volume of data exchanged. Although term distributed scheme has good efficiency for search, but it will result in load imbalance of the index servers. In this paper, to improve the retrieval efficiency and balance load, we takes load balanced term distribution strategy to build distributed index for secret documents.

6 Conclusions

We present Mimir, a distributed cipher retrieval system for sensitive documents. Mimir constructs the distributed indexes based on term distribution for storing the index in a load balanced way. Mimir takes encryption, key update, and access control to protect sensitive data, in which it encrypts the terms in indexes with random key for safety and efficiency, utilises partial key update to decrease the potential risk of attacks by malicious users, uses the access control based on role and user to control users to access the authorised data, and uses dynamic pipelined search strategy to balance the load of the management server and reduce the search delay. Our experiments show that Mimir can effectively protect secret data and answer queries almost as fast as an ordinary inverted index.

Currently, the index scale of Mimir is three million documents, and the search can be achieved on average 0.3 second. Our objective is to support larger data collection with similar search response. A challenging extension is to improve the scalability of term distribution in Mimir. Increasing new index servers will lead to changing the structure of the distributed cipher index. By doing so, the entire index needs to be reconstructed, which has a very large overhead. Another interesting question is how to protect the information of the posting list without encrypting it as whole, which can guarantee not to leak probabilistic information without sacrificing retrieval efficiency.

Acknowledgements

This work is supported by National Natural Science Foundation of China under Grants 61173170 and 61300222, National High Technology Research and Development Program of China under Grant 2007AA01Z403, Innovation Fund of Huazhong University of Science and Technology under Grants 2012TS052, 2012TS053 and 2013QN120, Natural Science Foundation of Hubei province of China under Grants 2013CFB310, and science and technology research project of the education department of Hubei province under grant B2013205.
References


