Mimir: Term-Distributed Indexing and Search for Secret Documents

Guoqiang Gao, Ruixuan Li, Xiwu Gu, Kunmei Wen, Zhengding Lu, Kun Yan
Intelligent and Distributed Computing Lab, School of Computer Science and Technology
Huazhong University of Science and Technology, Wuhan 430074, P. R. China
Email:ggq@smail.hust.edu.cn, {rxli, xwgu, kmwen, zdlu}@hust.edu.cn, yankun@gmail.com

***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) without imposing large load on the receptionist, and (3) 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 utilizes encryption with random key, partial key update, and access control based on role and user to protect sensitive data and improve query efficiency. Mimir 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 nearly as fast as an ordinary inverted index.**

***Index Terms*—ciphertext retrieval system, index, search, term distribution, encryption.**

I. 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 is a good choice [1]. The inverted index consists of a couple of data structures, namely lexicon and posting lists. The former stores all the distinct terms contained in documents. The latter is an array of list storing terms occurrences within the document collection. Current search engines mostly take the inverted index to store information, such as Google [2]. In order to prevent leakage of sensitive information, access control is the most widely used way. We take the access control based on role and user to prevent unauthorized users from accessing information. 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. Although encrypting the index can prevent data from being illegally stolen, however, it will lead to large computational overhead. In order to obtain high efficiency, only the lexicon needs to be encrypted because the document can be only derived from lexicon in the index. 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 secret documents stored in some intranets reaches at the terabyte range. If this volume of data were stored and indexed on a single computer, queries would 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 and term-distributed index. Each processing server stores the index corresponding to a subset of the documents in a document-distributed system. Queries are 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 [3]. 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 search system usually consists of the following components: client, receptionist and query processor. 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 of the system. 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 the pertinent index servers, and processes this information centrally. This strategy has the drawback of a severe bottleneck at the receptionist. An alternative is the pipelined term-distributed evaluation strategy proposed by Moffat et al. [4], where the query processing is distributed across the index

servers. However, pipelined search will increase the search delay because the search is done on the index servers one by one. We propose the dynamic pipelined search to trade off the bottleneck and the delay. If the communication load of the management server is high, we take full pipelined search to decrease the load, namely the search request is routed directly to the index servers and then the search is done on the index servers linearly. If the load of the management server is low, the search request is divided into multiple sub-queries and forwarded to pertinent index servers.

To address these problems, we propose Mimir[1], a distributed retrieval system for secret documents. In order to improve security, Mimir uses the cipher index and the access control 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, and achieves the distributed query based on dynamic pipelined search strategy to improve the load of the management server and the search delay. Mimir not only guarantees the safety of secret documents, but also the efficiency of the retrieval system.

This paper is organized as follows. Related work is presented in section 2. In section 3, we describe Mimir in details from load balanced term distribution, random key, partial key update, access control and dynamic pipelined search. Section 4 evaluates the performance through experiments. We conclude and summarize the results in section 5.

II. Related Work

Index security has been addressed by many researches, where the goal is to secure the secret documents from unauthorized access. Typically a system must provide two ways to achieve security: access control and data encryption [5]. Of the two, access control is a relatively older way to protect sensitive data. Currently, the access control for documents usually is discretionary access control (DAC) strategy [6] or role-based access control (RBAC) [7], [8]. Zerber [9] proposed a based user-group access control for index, in which a user can belong to multiple groups, but a document can only be owned by one group. This strategy is not very flexible because the user must belong to the group containing the document if he/she wants to access the document. Mimir uses the access control based on role and user, in which the roles and users can be authorized at the same time, to increase system flexibility.

Encryption is a standard technique for storing data confidentially [10]. Bertino [11] provides a framework for policy-based protection of XML data by encryption. Seitz [12] proposed an architecture that allows users to store and share encrypted data in the grid computing environment. Each posting list element in Zerber [9] 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. Margaritis [13] 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 [14] 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 partial key update, Mimir builds secondary index for the terms which 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. [15] 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 categorized into two types: document distributed and term distributed schemes. Harman et al. [16] described a document distributed system that was successfully deployed in practice. Cahoon et al. [17] 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 [18], in which the cluster of servers maintain 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, Ricardo A. Baeza-Yates et al [19] 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. [20]

[1]Mimir is the giant in Norse mythology who guards the "Well of the Highest Wisdom", situated in Jotunheim under of the roots of Yggdrasil, the World Tree.

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. Webber et al. [4] showed that term partitioning resulted in lower utilization 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.

In the case of the term distributed system, there is an evident lack of balance in the distribution on the load of the management server. In this paper, we use the terms of "management server" and "receptionist" interchangeably. To solve this problem, Xi et al. [21] proposed a hybrid method, where each inverted list is broken into k fixed-size chunks and one chunk is held on each node. However, this strategy can not utilize the advantages of document distribution and term distribution. An approach to eliminate the bottleneck of the management server is to use pipelining [22]. In this approach, the query is evaluated in stage by the sequence of servers that hold the inverted lists corresponding to the query terms. Evaluation of the query begins on first index server, which processes the posting lists corresponding to query terms to produce a set of accumulators. This set is passed to the next index server, which processes the lists for query terms against these accumulators to produce a modified set. The rest can be done in the same manner. Finally the last index server produces a final set of accumulators and returns it to the management server. However, the pipelined search will increase the delay of search. In this paper, We propose the dynamic pipelined search to trade off the bottleneck and the delay.

III. Mimir Distributed Cipher Retrieval System

Mimir is a distributed retrieval system for secret documents, which has two main functions: distributed indexing and distributed search. We will discuss each feature of Mimir in details after an overview of the architecture.

The components of Mimir include the clients, the management server, the index servers, the document servers, the random key database (RKDB) and the encryption server as shown in Figure 1. In the client (web browser), an authorized user can log onto the management server to build an index or submit a query. The management server distributes the entire index to *p* index servers as well as *r* secret documents to *q* document 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. A document is parsed into a series of terms which 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 called cipher index. All the cipher indexes stored in the index servers

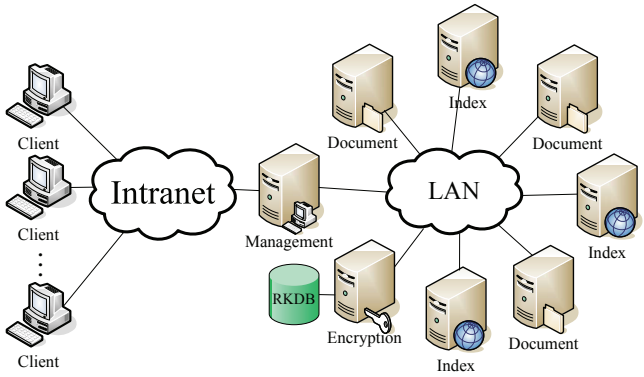

Fig. 1. The architecture of Mimir.

constitute the distributed cipher index which is complete logically. The index server is also responsible for handling queries, so it also can be called the query server. The query is parsed into multiple search 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. In the indexing and querying processes, the terms are encrypted through the encryption server, and a term obtains the key stored in RKDB through its hash value. In the front end, Mimir sets a flexible access control policy to protect sensitive data from unauthorized access; in the back end, encryption takes care of data leakage from illegal use.

A. Load Balanced Term distribution

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_t which are assigned into each index server in term distribution is refined in equation (1).

$$s_t = \frac{m}{p} \tag{1}$$

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 $T_A = (t_1, t_2, ..., t_{s_t})$, 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).

$$f(I_A) = \sum_{t_i \in T_A} f(t_i) \tag{2}$$

Since the frequencies of terms are different, according to Zipf's law, the term frequency of each index server is also different even with the same number of terms. Higher frequency of a term results in more documents containing the term whose posting list may be a bit longer. If $f(I_A) > f(I_B)$, the index size of server I_A is bigger than the size of server I_B. In the query, the load of index server I_A will be greater than the load of index server I_B.

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 $L(term, index\ server)$ 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 $f(I)$ of the index server *I* by $f(I) += f(t)$. This is presented in Algorithm 1.

Algorithm 1 Load Balanced Term distribution (term *t*)

```
if f(t) > f_threshold then
  if t not in L then
    I = {I | 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 = hash(t) mod p
  allocate t to index server I
end if
```

B. Random Key

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.

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 cipertext 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 which 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.

TABLE I
THE DATA STRUCTURE FOR KEY

SN	Key	Temp Key	Terms
0	key0		computer;...
1	key1		cipher;index;...
...	...	...	...

RKDB has two functions: to store the key and protect the key, as a result, RKDB must be an encrypted database. Table I 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 the 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 1031, and *M* is 5000, thus the key for "Chinese" can obtain from RKDB through the remainder 1031. 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.

Algorithm 2 Obtain Key (term *t*)

```
sn = hash(t) mod M
key = RKDB[sn].Key
if t not in RKDB[sn].Terms then
  add t to RKDB[sn].Terms
end if
return key
```

C. Partial Key Update

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. Random partial key updating can disrupt the correspondence between term and key, thereby, enhancing the security of the

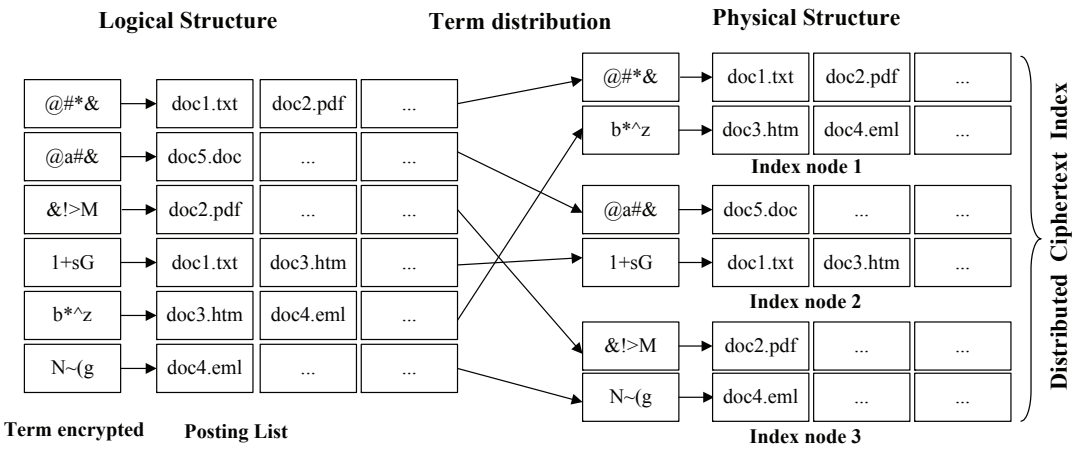

Fig. 2. The structure of the distributed cipher index with six terms.

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 this key 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 can not 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).

$$\sum_{i=2}^{n}(i-1)=\frac{n(n-1)}{2} \tag{3}$$

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 come 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.

D. Access Control in Mimir

To answer user queries, Mimir enforces access control based on role and user on its posting elements. Upon query,

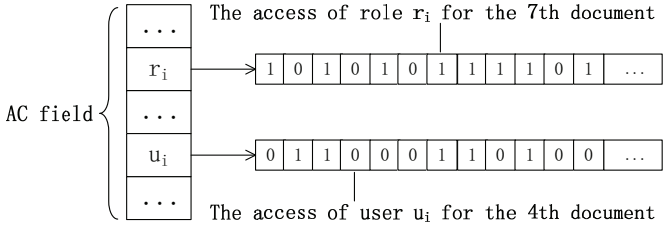

Fig. 3. Bitmap storage structure of ACI.

the management server authenticates the user and determines the roles to which the user belongs. For this purpose, the management server records which roles a user belonged to, and Mimir builds the access control index (ACI) to record which documents in posting list are accessible to each role or user. the ACI as in Figure 3 is stored in the management server and encrypted as a whole. When Mimir is initiated, the ACI is decrypted and placed into memory. The posting list of ACI forms the access control bitmap for the indexed documents. The term of ACI includes roles and users, and the corresponding posting list contains *n* bits, where *n* is the number of documents. The "1" in Figure 3 indicates the role or user can access the corresponding document and the "0" can not. Mimir can set access permission for role as well as user, which increases system availability and flexibility.

Suppose user *u* has a role set $R=(r_0,...,r_i)$, and query *q* can be parsed into a term set $T=(t_0,...,t_j)$. Let,

S_{t_i} is the result set of query term t_i.

S_{r_i} is the result set of query role r_i.

S_{u_i} is the result set of query user u_i.

The search authority set $S_{AC(u,R)}$ for user *u* is defined in equation (4).

$$S_{AC(u,R)}=S_u\cup\left(\bigcup_{k=0}^{i}S_{r_k}\right)\cap S_u \tag{4}$$

If there is *u* in ACL, we filter the search results only according to S_u, otherwise, based on S_r where user *u* has role *r*.

The result set S_q for query q is defined in equation (5).

$$S_q = \bigcup_{k=0}^{j} \left(S_{t_k} \cap S_{AC(u,R)}\right) \tag{5}$$

For example, a user u has role set r_1, r_3, the query q is "Beijing Olympic". The query result is refined as following:

$$\begin{aligned} &(S_{Beijing} \cap (S_u \cup S_{r_1} \cup S_{r_3} \cap S_u)) \\ &\cup (S_{Olympic} \cap (S_u \cup S_{r_1} \cup S_{r_3} \cap S_u)) \end{aligned} \tag{6}$$

E. Dynamic Pipelined Search

Distributing the tasks of an information retrieval system enables a number of desirable features, such as improving the search efficiency. A major drawback that arises from the distribution of tasks across a number of index servers is the communication among these servers. Especially, all index servers need to communicate with the single management server and the retrieval results produced in the index servers always have a huge scale. So the management server can be a bottleneck as a large number of users simultaneously use the system. Although the pipelined search which does search in one-by-one mode can relieve the communication load on the management server, it increases the retrieval delay as well, and the query must bear the same search latency even in lower load network.

In this paper, we dynamically adjust the size of pipelined search based on the traffic load of the management server. In search, the retrieval words are analyzed into multiple search terms, and there is multiple index servers corresponding to these search terms. Let search group $S_i = \{s_i, (t_{i1}, t_{i2}, ..., t_{in})\}$ to express that the terms $(t_{i1}, t_{i2}, ..., t_{in})$ should be searched in the index server s_i. Suppose that query Q is parsed into n search groups $(S_1, S_2, ...S_n)$. If the load of the management server is more than 90% of full load, Mimir take full pipelined search to relieve the load of management server. That is the management server sends search group $(S_1, S_2, ...S_n)$ to the index server s_1 to search the terms $(t_{11}, t_{12}, ..., t_{1n})$, then the index server s_1 sends the remaining search group $(S_2, ...S_n)$ to the index server s_2 to search the terms $(t_{21}, t_{22}, ..., t_{2n})$ and the index server s_2 merges its results with the results from the previous index server, and so on. In this way, the management server has low communication load because it only receives the results from the last index server, however, this full pipelined search increases search delay. If the load of the management server is about 60% of full load, the management server divides the search group in half and runs two pipelined search. If the load of the management server is less than 30% of full load, the management server does not take pipelined search strategy and sends the query to the index servers $(s_1, s_2, ...s_n)$ to parallel search. The last approach has the best search delay and maximum communication overhead. According to dynamical pipelined search, we can obtain a better traffic load and delay.

F. Using Mimir

In this section, we will describe distributed indexing and search in Mimir.

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 document 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, and updates the access control index.

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 extracts the terms corresponding 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) Distributed Search: To execute a keyword query, the management server firstly authenticates the user. If the authentication is passed, the management server searches ACI based on the user u and its role set R, and obtains the search authority $S_{AC(u,R)}$ according to the equation (4). Processing queries in a distributed fashion consists of determining which resources to allocate from a distributed system when processing a particular query. Secondly, 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. Thirdly, the management server sends the cipher retrieval terms and their $S_{AC(u,R)}$ to the corresponding index servers to perform dynamic pipelined search based on its load. In the index servers, also can be called query servers as previous description, the cipher retrieval terms are searched over cipher index with filter condition $S_{AC(u,R)}$, and the result is merged with the result from the previous index server, and the merged result is returned to the next index server or the management server. Finally, the management server receives the search results from only one index server or multiple index servers, and shows the final results to the users. If the users want to view or download a document, the management server obtains the document from the document servers and sends it to the users. Compared to the document distribution, Mimir processes distributed search only over part of the index servers. Therefore, Mimir reduces the load on the

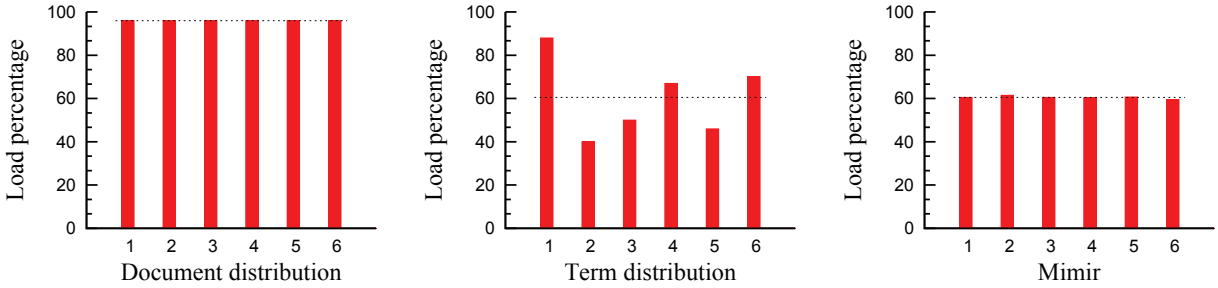

Fig. 4. Distribution of the average load per node in document distribution, term distribution and Mimir.

system. Since only the terms are encrypted, the search strategy of Mimir is the same with the plain retrieval engines through converting the plain term into the cipher term. 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 $f(t) > f_{threshold}$ **then**
  $lookup\ t\ in\ L\ and\ get\ I$;
else
  $I = hash(t)\ mod\ p$
end if
$return\ I$

IV. 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, using the Reuter Corpus whose volume is 1.55G. The hardware used in all the experiments described in this section is a Beowulf-style cluster of 7 computers, each a 2.5GHz Intel Xeon with 2GB RAM and 1TB local SATA disk in a RAID-5 configuration. One of the cluster 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 both the index servers and the document servers.

A. 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 stored in document servers. 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, the access control schemes based role and user are put in place to protect the documents contents from unauthorized 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.

B. 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 6 index servers of a document distributed system (left), a term distributed system (middle) and Mimir (right). 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

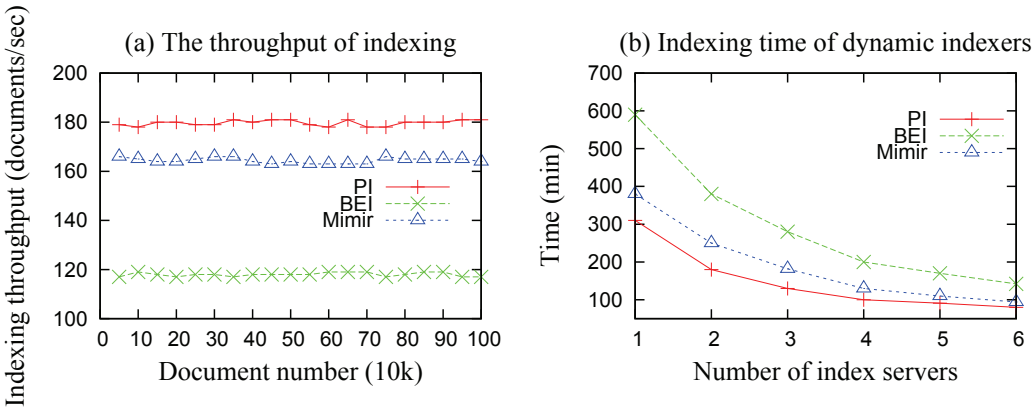

Fig. 5. Indexing performance.

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.

C. Indexing Performance

In order to discuss the efficiency of the index build, we analyze 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. We take the 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 5(a) shows the throughput of indexing with three different strategies. 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(b) shows the indexing time of dynamic index servers with three different strategies for the collection of Reuter Corpus. It is clear that as the number of dynamic index servers increases, indexing time decreases. Mimir needs about one hundred 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.

D. Query Performance

We evaluate effectiveness of query processing by analyzing latency of queries in which query latency is represented by the average response time required to process a query. Figure 6(a) represents the query response time with Mimir, PI and BEI. 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.

The dynamical pipelined search (DPS) in Mimir trades off the load of the management server and the delay of search. In order to analyze the efficiency of DPS, we let Mimir take the pipelined approach (PA) proposed by Moffat [22] as search strategy to compare performance. Figure 6(b) plots the average search delay over 500 queries at a time for Mimir and PA. When there are a lot of concurrent queries, the search delay of Mimir and PA is similar because the pipelined search of Mimir is the same with PA. However, when the management server is not busy, Mimir do not take pipeline search strategy to decrease the search delay. Therefore, Mimir can improve system efficiency compared to PA.

V. Conclusion & Future Work

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, it utilizes partial key update to decrease the potential risk of attacks by malicious users, it uses the access control based on role and user to control users to access the authorized data. It uses dynamic pipelined search strategy to balance the load of the management server and reduce the search delay. Our

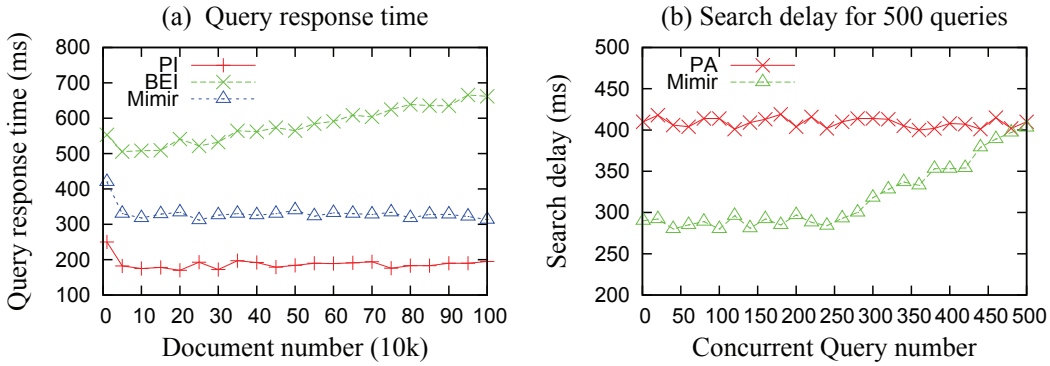

Fig. 6. Query performance.

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.

VI. Acknowledgments

This work is supported by National Natural Science Foundation of China under Grant 60873225, 60773191, 70771043, National High Technology Research and Development Program of China under Grant 2007AA01Z403, Natural Science Foundation of Hubei Province under Grant 2009CDB298, Wuhan Youth Science and Technology Chenguang Program under Grant 200950431171, Open Foundation of State Key Laboratory of Software Engineering under Grant SKLSE20080718, and Innovation Fund of Huazhong University of Science and Technology under Grant Q2009021.

References

[1] R. A. Baeza-Yates and B. A. Ribeiro-Neto, *Modern Information Retrieval*. ACM Press / Addison-Wesley, 1999.

[2] Google, "http://www.google.com."

[3] C. S. Badue, R. A. Baeza-Yates, B. A. Ribeiro-Neto, A. Ziviani, and N. Ziviani, "Analyzing imbalance among homogeneous index servers in a web search system," *Inf. Process. Manage.*, vol. 43, no. 3, pp. 592–608, 2007.

[4] A. Moffat, W. Webber, J. Zobel, and R. A. Baeza-Yates, "A pipelined architecture for distributed text query evaluation," *Inf. Retr.*, vol. 10, no. 3, pp. 205–231, 2007.

[5] T. Ge and S. B. Zdonik, "Fast, secure encryption for indexing in a column-oriented dbms," in *Proc. of the 23rd International Conference on Data Engineering*, 2007, pp. 676–685.

[6] E. Damiani, S. D. C. di Vimercati, S. Paraboschi, and P. Samarati, "A fine-grained access control system for xml documents," *ACM Trans. Inf. Syst. Secur.*, vol. 5, no. 2, pp. 169–202, 2002.

[7] R. Chandramouli, "Application of xml tools for enterprise-wide rbac implementation tasks," in *Proc. of the 5th ACM Workshop on Role-Based Access Control*, 2000, pp. 11–18.

[8] J. Hu, R. Li, and Z. Lu, "Rbac-based secure interoperation using constraint logic programming," in *Proc. of the 12th IEEE International Conference on Computational Science and Engineering*, 2009, pp. 867–872.

[9] S. Zerr, E. Demidova, D. Olmedilla, W. Nejdl, M. Winslett, and S. Mitra, "Zerber: r-confidential indexing for distributed documents," in *Proc. of the 11th International Conference on Extending Database Technology*, 2008, pp. 287–298.

[10] M. Kallahalla, E. Riedel, R. Swaminathan, Q. Wang, and K. Fu, "Plutus: Scalable secure file sharing on untrusted storage," in *Proc. of the FAST'03 Conference on File and Storage Technologies*, 2003.

[11] E. Bertino, S. Castano, and E. Ferrari, "Securing xml documents with author-x," *IEEE Internet Computing*, vol. 5, no. 3, pp. 21–, 2001.

[12] L. Seitz, J.-M. Pierson, and L. Brunie, "Key management for encrypted data storage in distributed systems," in *IEEE Security in Storage Workshop*, 2003, pp. 20–30.

[13] G. Margaritis and S. V. Anastasiadis, "Low-cost management of inverted files for online full-text search," in *Proc. of the 18th ACM Conference on Information and Knowledge Management*, 2009, pp. 455–464.

[14] S. Gurajada and P. S. Kumar, "On-line index maintenance using horizontal partitioning," in *Proc. of the 18th ACM Conference on Information and Knowledge Management*, 2009, pp. 435–444.

[15] J. Li, Q. Wang, C. Wang, N. Cao, K. Ren, and W. Lou, "Fuzzy keyword search over encrypted data in cloud computing," in *Proc. of IEEE INFOCOM 2010 mini-conference*, 2010.

[16] D. Harman, W. McCoy, R. Toense, and G. Candela, "Prototyping a distributed information retrieval system that uses statistical ranking," *Inf. Process. Manage.*, vol. 27, no. 5, pp. 449–460, 1991.

[17] B. Cahoon, K. S. McKinley, and Z. Lu, "Evaluating the performance of distributed architectures for information retrieval using a variety of workloads," *ACM Trans. Inf. Syst.*, vol. 18, no. 1, pp. 1–43, 2000.

[18] L. A. Barroso, J. Dean, and U. Hölzle, "Web search for a planet: The google cluster architecture," *IEEE Micro*, vol. 23, no. 2, pp. 22–28, 2003.

[19] R. A. Baeza-Yates, C. Castillo, F. Junqueira, V. Plachouras, and F. Silvestri, "Challenges on distributed web retrieval," in *Proc. of the 23rd International Conference on Data Engineering*, 2007, pp. 6–20.

[20] A. MacFarlane, J. A. McCann, and S. E. Robertson, "Parallel search using partitioned inverted files," in *Proc. of the Seventh International Symposium on String Processing Information Retrieval*, 2000, pp. 209–220.

[21] W. Xi, O. Sornil, and E. A. Fox, "Hybrid partition inverted files for largescale digital libraries," in *Proc. of Digital Library: IT Opportunities and Challenges in the New Millennium*, 2002, pp. 401–418.

[22] A. Moffat, W. Webber, and J. Zobel, "Load balancing for term-distributed parallel retrieval," in *Proc. of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2006, pp. 348–355.