Uniform Storage Model-based Update Scheme of On-line Information Retrieval System

Xiaozhu Liu¹,*, Yuwei Peng ², Yuanhua He ¹, and Zukai Tang ³

1. School of Automation, Wuhan University of Technology, Wuhan 430070, China
2. School of Computer, Wuhan University, Wuhan 430072, China
3. School of Computer Science and Technology, Wuhan University, Wuhan 430070, China

*Corresponding author, e-mail: lxz_h@163.com

Abstract—In order to improve the retrieval performance of on-line information retrieval systems, an efficient index update scheme is proposed in this paper, which can provide better skipping function and further enhance both space and time efficiencies without inserting any additional auxiliary information. A uniform storage model (USM) is proposed to manage both short and long postings lists based on link. A USM-based update scheme also is proposed to distinguish long and short posting lists, which merges short lists with immediately merge, and merges long lists with improved Y-limited contiguous multiple merge scheme, which balances the trade-off of the time and space efficiencies effectively. The proposed update scheme not only considers both index level and inverted list level update, but also effectively improves time and space efficiencies of index update. Detailed experimental results and comparison with existed schemes show that the proposed scheme greatly averagely reduces space cost, conjunctive Boolean query time, and the cost of on-line index construction.

Index Terms—Uniform Storage Model, Information Retrieval, Update Scheme, Inverted Index

I. INTRODUCTION

On-line information retrieval system is a key and challenge technology to manage and search information efficiently [1, 2]. Inverted index [3], as one of efficient index files for on-line information retrieval, has been comprehensively studied in recent years [4, 5]. Although compression schemes [6, 7] can greatly reduce disk access time, the compressed index for each query term must be completely decompressed, which will degrade query performance to some extent, especially for huge amount of text. On the other hand, index update strategies in dynamic search environments also have great influence on the space and time performance of on-line information retrieval systems [8, 9]. Hence, design an efficient index update scheme to improve time and space performance is an important and challenge task for on-line retrieval systems.

Existed work shows that the self-index [9, 10] is a promising way to improve retrieval system performance by compressing inverted index, which employ skipping mechanism to provide fast addressing function with inserting some additional auxiliary information. And many on-line index update schemes [11, 12, 13, 14] such as immediate merge maintenance, generation-based LOG merge, dynamic balancing tree-based update and hybrid update schemes are proposed to improve space and time efficiency. Although such research works have explored to improve update performance of on-line retrieval system, there are several problems for the existed mechanisms: 1) existed mechanisms cannot manage index file efficiently, which will incur high storage overheads. And the increase in disk I/O time outweighs the reduction in decompression time for huge amount of data; 2) the existed index update strategies just focus on the index file level, which didn’t take the posting list structure into consideration, especially for long postings and short postings. After all, designing an efficient index update scheme is still an important and challenge research field worthy of studying.

Focusing on improving space and time performance of on-line information retrieval, an efficient index update scheme is proposed in this paper, which follows our previous random access blocked inverted (RABI) index [10] to provide better skipping function and further enhance both space and time efficiencies without inserting any additional auxiliary information. Based on RABI and Zipfian theorem [15], a uniform storage model (USM) is proposed to manage both short and long postings, which manages short and long lists with uniform storage model of distinguishing long and short lists based on link. A USM-based update scheme also is proposed to distinguish between long and short posting lists, which merges short lists with immediately merge, and merges long lists with improved Y-limited contiguous multiple merge scheme, which balances the trade-off of the time and space efficiencies effectively. The proposed scheme not only considers both index level and inverted list level update, but also effectively improves time and space efficiencies of index update.

The rest of this paper is organized as follows. An efficient uniform storage model (USM) and USM-based update scheme are proposed in Section 2. Detailed experimental results are shown in Section 3. The paper concludes with Section 4.
II. PROPOSED USM-BASED UPDATE SCHEME

A. Proposed Uniform Storage Model

Inverted index is the fundamental data structure employed by most search engines. In order to overcome the problems of the huge space cost, low query performance and unable to support both conjunctive Boolean query and ranking query simultaneously of inverted index, the main idea of the proposed uniform storage model (USM) is based on our previous proposed random access blocked inverted index (RABI) [9], which appropriately divides inverted list into sub-blocks, and then compress different part of each sub-block with the corresponding compression method, which makes fast location and random access of compressed index into reality without inserting any additional auxiliary information. The structure of RABI is shown as in Figure 1.

![Figure 1. Structure of random access blocked inverted index.](image)

In Fig. 1, RABI is made up of two parts: words table and posting lists. Words table \( w_i \) includes the word \( w_i \), \( s_i \) the number of documents, and pointer \( p_i \), which points to the location of the blocked posting list \( L_{i, \text{block}} \) of \( w_i \). Each blocked posting list \( L_{i, \text{block}} \) is made up of \( m \) sub-blocks \( S_{\text{block}}^r \) ( \( r \in [1,m] \)). Every sub-block \( S_{\text{block}}^r \) includes two sections: locating section \( L_{\text{Loc}} \) and information section \( I_{\text{F}} \). Locating section \( L_{\text{Loc}} \), which is compressed with coding method \( C_{\text{Loc}} \), implements fast locating of compressed postings lists and sub-block level random access without decoding the compressed lists. \( L_{\text{Loc}} \) is the pair of the first document ID and the corresponding cumulative within-document frequency. Information section \( I_{\text{F}} \), which is compressed with efficient compact binary coding method \( C_{\text{F}} \) without inserting any additional auxiliary information, implements fast locating of compressed inner sub-block and posting level random access with partial decoding the compressed sub-block. For the information sections, except the information section \( I_w \) that is in the last sub-block \( S_{\text{block}}^m \), is the residual postings, other information section \( I_{\text{F}} \) is made up of \( k-1 \) lists \( L_{\text{F}} \) and \( k-1 \) lists \( L_{\text{L}} \), where \( L_{\text{F}} \) is the ascending list of document IDs, and \( L_{\text{L}} \) is the ascending list of cumulative within-document frequency.

According to the Zipfian distribution [15], more than 50% inverted lists are short list, and the size is less than 32 byte. Moreover, with the increasing of text set, the length of the inverted list of these words does not increase much. The amount of long inverted lists is only about 10% of the entire dictionary, and the long inverted lists of these words constitute the main body of the entire inverted index, which means that these long inverted files occupy the most storage space. In addition, the long inverted lists will increase obviously with increasing of the documentation set.

Sorting the frequency of term set \( \{W\} \) in natural language documentation set \( D \), for the \( m \)-th most frequent term \( W_m \), according to the Zipfian distribution [15], the collection frequency \( f(W_m) \) (the number of times the term appears in the collection \( D \)) meets:

\[
f(W_m) = C / m^\theta,
\]

where \( C \) and \( \theta \) are collection-specific constants.

Let \( N \) be the total number of terms, \( M \) be the total number of times the term appears in the collection \( D \). Then we have:

\[
\sum_{m=1}^{N} f(W_m) = M.
\]

According to expression (2), we can get the probability \( p(W_m) \) of term \( W_m \) in the collection \( D \):

\[
p(W_m) = f(W_m) / M = C / (M \cdot m^\theta).
\]

According to the rule of probability distributions, it is clear that the probability \( p(W_m) \) satisfy the following constraints:

\[
\sum_{m=1}^{N} p(W_m) = 1.
\]

Take expressions (1) and (3) into (4), we have:

\[
\frac{C}{M} \sum_{m=1}^{N} m^{-\theta} = 1,
\]

in (5), considering \( N \) tends to infinity, namely the total number of terms in document collection tends to infinity, with \( \theta > 1 \), we have:

\[
\lim_{N \to \infty} \frac{C}{M} \sum_{m=1}^{N} m^{-\theta} \approx \frac{C}{M} \left( \frac{1}{\theta-1} - \lim_{N \to \infty} (\ln N - \sum_{m=1}^{N} m^{-1}) \right) = 1,
\]

further simplify (6), we can get:

\[
C = \frac{M}{(\theta-1)^{-1} - \lim_{N \to \infty} (\ln N - \sum_{m=1}^{N} m^{-1})}.
\]
take expression (7) into (1), we have:

\[
f(W_n) = \frac{M}{m^\theta \cdot [(\theta - 1)^{-1} - \lim_{N \to \infty} (\ln N - \sum_{m=1}^{\infty} m^{-1})]} \approx \frac{M}{m^\theta \cdot [(\theta - 1)^{-1} - 0.5772]},
\]

(8)

For the document collection \(D\) in actual application system, \(M\), the total number of times that appears in \(D\), is finite. Hence, the actual number of words and word frequency that appears in \(D\) are a subset of the above derivation of results in expression (8). According to Heaps law [10], the expected value \(E(M')\) of the actual total number \(M'\) of words in the document is:

\[
E(M') = \sum_{m=1}^{\infty} \left( 1 - \left( 1 - (\theta - 1)^{-1} - 0.5772 \right) m^\theta \right)^m.
\]

(9)

simplify (9), we can get the actual total number \(M'\):

\[
M' = \left( \frac{2}{(\theta - 1)^{-1} - 0.5772} \right) ^{\frac{1}{\theta}} \cdot M^{\frac{1}{\theta}}.
\]

(10)

The expression (10) indicates that the actual total number \(M'\) will not increase linearly with increasing of the number of documents, but increase relatively slowly, which is related to constant \(\theta\) of documentation set.

Considering that short lists have high merging efficiency and long lists have low merging efficiency in the process of updating and maintenance. In the proposed uniform storage model (USM), we distinguish the lists with the length of the inverted list. Let \(X\) be the threshold of inverted list. If the length of posting list is larger than \(X\), the list will join in the long posting lists.

In order to overcome the defects of managing long and short inverted list with different data structures in existed methods, and improve query efficiency, in the proposed USM, the unified storage model based on the link is employed to manage both short and long lists. For short lists, the continuous block index technology is used. And link list with dynamic pre-allocated space is adopted to store a long list.

According to the Zipfian distribution [15], and considering the length of the short list is generally very short, the requirement of new space is generally small during the data increasing. In order to avoid the space overhead, while improving the efficiency of dynamic update, the continuous sub-block technology is employed to manage short lists, which means that the continuous physical space will be divided into many blocks with the same size \(S_i\). And the management structure of the short list is shown as in Figure 2.

In Figure 2, words table \(W_i\) includes the word \(w_i\), the number \(s_i\) of documents, and pointer \(p_i\) which points to the location of the blocked posting list. And all short lists are allocated the same size continuous physical space \(S_i\) to reduce the number of disk operations generated in the dynamic update process.

Figure 2. Management structure of short list.

Long list is the length of posting list is larger than \(X\), and the management structure of long list is shown as in Fig. 3. In the proposed management structure, pointer \(p_i\) which points to the location of the blocked posting list is also stored in the word vocabulary. Since the space occupied by long list is large, all long lists are allocated with non-contiguous physical space with different size. And the space will be allocated dynamically according to the incremental data, which will minimize the number of disk I/O operations generated in the dynamic update process.

In order to evaluate the number of short lists, the proposed USM will consider the more general case where lists include the posts information. According to the definitions of long list and short list, the long lists satisfy the condition:

\[
f(W_m) > X,
\]

(11)

take expression (8) into (11), we can obtain the number \(N_L\) of long lists:

\[
N_L = \left( \frac{M}{X \cdot (\theta - 1)^{-1} - 0.5772} \right)^{\frac{1}{\theta}}.
\]

(12)

Then we can get the number \(N_S\) of short lists:

\[
N_S = \left( \frac{2}{(\theta - 1)^{-1} - 0.5772} \right)^{\frac{1}{\theta}} \cdot M^{\frac{1}{\theta}} - \left( \frac{M}{X \cdot (\theta - 1)^{-1} - 0.5772} \right)^{\frac{1}{\theta}}
\]

(13)
With the number \( N_L \) of long lists and the number of frequency term, we can obtain the number \( N_{L,P} \) of posts in long lists \( N_{L,P} \):

\[
N_{L,P} = \frac{N_L}{f(W_p)} = X^{\frac{1}{\beta}} \frac{1}{\theta^\frac{1}{\beta}} \cdot \frac{1}{[(\theta - 1)^{\frac{1}{\beta}} - 0.5772]^{\frac{1}{\beta}}},
\]

(14)

Similarly, we can get the number \( N_{S,P} \) of post in short lists:

\[
N_{S,P} = \frac{X^{\frac{1}{\beta}} \frac{1}{\theta^\frac{1}{\beta}} \cdot \frac{1}{[(\theta - 1)^{\frac{1}{\beta}} - 0.5772]^{\frac{1}{\beta}}}}{[(\theta - 1)^{\frac{1}{\beta}} - 0.5772]^{\frac{1}{\beta}}},
\]

(15)

\[ \text{B. Proposed Update Scheme} \]

Considering that long posting lists take more time to copy than the whole lists to perform a single disk seek operation, while for short lists a disk seek is more expensive than copying the list as part of a longer, sequential read/write operation, the proposed uniform storage model-based update scheme (USMUS) will distinguish long posting lists and short posting lists, and update them with the corresponding method. For a given threshold \( X \), if the length of posting list is larger than \( X \), the list will join in the long posting lists. For the new posting list \( L_N \) of word \( w_{n_p} \), the proposed USMUS is shown as in algorithm 1.

\[ \text{Algorithm 1: USMUS} \]

**Input:** \( L_N, X, w_p, L_{short}, w_{n_p} \)

**Output:** updated \( w_{i_p} \) and \( L_{short} \)

1: if \( L_n < X \) 
2: if there is not the short list \( L_{short} \) of \( w_{n_p} \) in the index 
3: write \( L_n \) on the disk directly, and update \( w_{i_p} \) 
4: else 
5: Re-merge \( L_{short} \) and \( L_n \) resulting in a new list \( L' \) 
6: if \( L' < X \) 
7: Write \( L' \) on the disk directly and update \( w_{i_p} \) 
8: else 
9: Put \( L' \) into the long lists \( L_{n} \) and perform LLUA 
10: else 
11: Put \( L_n \) into the long lists \( L_{n} \) and perform LLUA 
12:return the updated \( w_{i_p} \) and \( L_{short} \)

<table>
<thead>
<tr>
<th>Algorithm 2: LLUA</th>
</tr>
</thead>
</table>
| **Input:** \( L_y, Y, w_i, L_{short}, w_{n_p} \), the new long list \( L_{n} \)
| **Output:** updated \( w_{i_p} \) and \( L_{short} \) |

1: if \( L_n < Y \) 
2: Insert it into \( L_i \) after \( L_i(K) \) and write it on the disk directly, and update \( w_{i_p} \) 
3: else 
4: Insert it into \( L_y(K) \) in descending order 
5: if \( L_y(j) \) meets the conditions: 
6: \( \sum_{j,K} \sum_{j,K} LE(j) < Y \). and \( W < M + 1 \) 
7: Re-merge the long lists \( L_y(j) \) \( K + 1 \leq j \leq W + 1 \) 
8: and update \( w_{i_p} \) 
9: return the updated \( w_{i_p} \) and \( L_{short} \)

\[ \text{Figure 4} \] shows an example of USMUS. When the threshold \( X \) is 5 and \( Y \) is 25. Initial long lists \( L_y \) includes four long lists, the length is 26, 10, 7, and 5 respectively. And the length of initial short list \( L_{short} \) is 4. For the new list \( L_N \) (length is 4), the proposed update scheme USMUS works as follows:

\[ \text{Initial long lists} \]

```
26 10 7 5
```

\[ \text{Initial short list} \]

```
4
```

1. Merge initial short list and the new list

```
4 4 8
```

2. Insert the new long list into Initial long lists according to length

```
26 10 8 7 5
```

3. Merge partial long lists

```
26 10 8 7 5 25
```

Short list

```
The long posting lists update algorithm (LLUA) is shown as in algorithm 2.
```

\[ \text{Algorithm 2: LLUA} \]

| **Input:** \( L_y, Y, w_i, L_{short}, w_{n_p} \), the new long list \( L_{n} \)
| **Output:** updated \( w_{i_p} \) and \( L_{short} \) |

1: if \( L_n < Y \) 
2: Insert it into \( L_i \) after \( L_i(K) \) and write it on the disk directly, and update \( w_{i_p} \) 
3: else 
4: Insert it into \( L_y(K) \) in descending order. 
5: if \( L_y(j) \) meets the conditions: 
6: \( \sum_{j,K} \sum_{j,K} LE(j) < Y \). and \( W < M + 1 \) 
7: Re-merge the long lists \( L_y(j) \) \( K + 1 \leq j \leq W + 1 \) 
8: and update \( w_{i_p} \) 
9: return the updated \( w_{i_p} \) and \( L_{short} \)

\[ \text{Figure 4} \] shows an example of USMUS. When the threshold \( X \) is 5 and \( Y \) is 25. Initial long lists \( L_y \) includes four long lists, the length is 26, 10, 7, and 5 respectively. And the length of initial short list \( L_{short} \) is 4. For the new list \( L_N \) (length is 4), the proposed update scheme USMUS works as follows:

\[ \text{Initial long lists} \]

```
26 10 7 5
```

\[ \text{Initial short list} \]

```
4
```

1. Merge initial short list and the new list

```
4 4 8
```

2. Insert the new long list into Initial long lists according to length

```
26 10 8 7 5
```

3. Merge partial long lists

```
26 10 8 7 5 25
```

Short list

```
```

In USMUS, since the length of \( L_N \) is 4 and there is an initial short list. So USMUS merges \( L_N \) and the initial list into a new list \( L' \) with length 8.

Then, considering that the length of \( L' \) is larger than the threshold \( X \). USMUS puts \( L' \) into the long lists \( L_{n} \) and performs LLUA. Then the short list becomes empty.
Because the length of $L'$ is less than the threshold $Y$ and is larger than the length of $L_2(3)$, LLUA inserts $L'$ into $L_2$ in descending order. Then LLUA will check the conditions:

$$\sum_{j-k+1}^{W} LE(j) < Y, \quad (18)$$

$$\sum_{j-k+1}^{W+1} LE(j) \geq Y, \quad (19)$$

$$W < M + 1. \quad (20)$$

Then we can find the long lists $L_2(j) \ (2 \leq j \leq 4)$ meet the above conditions. So LLUA merge the long lists and get the new long lists.

III. EXPERIMENTAL RESULTS AND DISCUSSION

A. Experimental Setup

To evaluate the efficiency of the proposed scheme, we used the standard (un-skipped) compressed inverted file as the baseline, in which d-gap and Golomb code with the appropriate parameter $b$ are used for each inverted list. This baseline is then used to evaluate self-index (SIF) [10] and the proposed USM file organizations that are implemented on the open source platform. All experiments were run on an Intel P4 3.0GHz PC with 1GB DDR memory system. The 7200-rpm SATA hard disk is 240GB, and the data transfer rate is 25MB/sec. The 127GB experimental data was provided by Wuhan Patent Office. Intervening processes and disk activities were minimized with best effort during experimentation.

The similarity calculations and document ranking, which form the major source of overhead in query processing. To reduce the CPU time and improves query performance, the asymptotic run-time costs for the creation, update, extraction and selection phases can be reduced with implementation optimization with efficient data structure and algorithm. Two important features in the inverted index structure let us devise a document-ordered query processing strategy. First, the postings of a term are stored in increasing order of document ids. That is, while traversing an inverted list, once a document id is seen in a posting, there cannot be a smaller document id in one of the succeeding postings in that list. Second, the number of query terms is limited. We have $Q$ terms to be processed. These observations allow us to process the inverted lists in parallel instead of processing them consecutively. This way, it is possible to compute a complete score for a document before all posting in the lists are completely processed. In this processing, update, extraction, and selection phases are performed in an interleaved manner.

B. Performance Analysis

The actual size ratio for each inverted file organization is shown in Figure 5. As expected, the size of the proposed USM is obviously less than that of SIF by 5.7% on average. With the number $k$ of postings per block increasing, the space costs of two schemes also decrease. This confirms that smaller blocks are inappropriate for both the SIF and the proposed USM inverted file. For SIF, the ratio to the baseline of standard compressed inverted file decreases as $k$ increasing. When $k$ increasing from 5 to 1025, the ratio decreases from 124.6% to 100.7%. The reason is that the less skipping information should be inserted in the inverted list with $k$ increasing, so the overhead will decrease. For the proposed USM file, its ratio decreases from 114.2% to 96.5%. Furthermore, when $k$ is 65, the space cost of USM is 98.9%, which implies that for the proposed USM file can incur no space overhead in creating the blocked index and decrease the space cost greatly. The results also show that the proposed USM provides a space economical way to implementing an efficient skipping inverted file.

In this scenario, we allowed the system to use 512 MB of main memory for the in-memory index. The size of the final index was 10.2 GB. To measure on-line index update and query processing performance at the same time, we simulated an on-line search environment and created a mixed update/search sequence consisting of 4586 update operations and 1487 search queries. To validate the proposed USMUS scheme, the simple hybrid maintenance scheme (SHMS, In-Place + Logarithmic Merge) in [12] and improved hybrid maintenance scheme (Hybrid Logarithmic Merge, HLMNC) in [13] were implemented as well. We analyzed the index maintenance and query processing performance using various parameters $X$ and fixed $Y = 10^7$. The experimental results are shown in Figures 6 and 7.

From the results shown in Fig. 6, it is obvious that the proposed USMUS significantly outperforms SHMS for its indexing performance. The total indexing time of USMUS is less than that of SHMS about 47.8% on average. The indexing performance of USMUS is close to that of HLMNC and is slightly better than that of HLMNC, when the threshold is $1 \times 10^6$ or $2 \times 10^6$. The reason is that the proposed USMUS merges short postings list and updates long posting lists by hybrid in-place and remerge strategy, which will reduce updating time and appropriately re-merging part of long posting lists can balance the maintenance performance.
Compared to SHMS (3.2 hours) and HLMNC (1.8 hours), USMUS with 512 MB RAM only needs 1.7 hours to build the final index when the threshold is $1 \times 10^6$.

![Index update performance for different strategies.](image)

![Query processing performance for different strategies.](image)

On the other hand, the query performance of USMUS also is the best in three schemes as shown in Fig. 7. And USMUS also exhibits a vastly superior query processing performance, namely 321 ms per query, which is obviously less than 347 ms (HLMNC) and 353ms (SHMS). The reason is that the thresholds $X$ and $Y$ can achieve a good tradeoff between maintenance performance and query performance. On the other hand, USMUS also considers the blocked structure of index, which can reduce query time by random access and partial decoding. So appropriate thresholds $X$ and $Y$ can improve both the index maintenance performance and query performance.

IV. CONCLUSION

To improve time and space performance of on-line information retrieval system, an efficient uniform storage model (USM) and USM-based update scheme (USMUS) are proposed, which manage short and long postings list with uniform link based on our previous self-index RABI to support random access function and optimize conjunctive Boolean and ranking queries. Compared with existed works, the proposed USMUS takes both index file and posting list structures into consideration and distinguishes short and long posting lists in storage space and update scheme. Detailed experimental results show that, compared with SIF, the proposed USMUS averagely reduces space cost by 5.6%, conjunctive Boolean query time by 17.8%. The experimental results also demonstrate that the proposed USMUS effectively reduces the cost of on-line index construction, which provides a very simple and attractive way to building a fast and space-economical on-line information retrieval system. The future work will focus on space management scheme to optimize timely retrieval performance.

ACKNOWLEDGMENTS

This work was supported in part by Foundation of State Key Laboratory of Software Engineering (SKLSE) of Wuhan University (No. SKLSE2010-08-09), National Natural Science Foundation of China (No. 61103019), the Fundamental Research Funds for the Central Universities of Wuhan University of Technology (No. 2012IV-101), and the Foundation of Young Teacher Join Enterprise Action Plan of Colleges of Hubei Province (No. XD2012020).

REFERENCES


