DOI: 10.2298/CSIS100407025J

Indexing Temporal Information for Web Pages

Peiquan Jin, Hong Chen, Xujian Zhao, Xiaowen Li, and Lihua Yue

School of Computer Science and Technology,
University of Science and Technology of China, 230027, Hefei, China
jpq@ustc.edu.cn

Abstract. Temporal information plays important roles in Web search, as Web pages intrinsically involve crawled time and most Web pages contain time keywords in their content. How to integrate temporal information in Web search engines has been a research focus in recent years, among which some key issues such as temporal-textual indexing and temporal information extraction have to be first studied. In this paper, we first present a framework of temporal-textual Web search engine. And then, we concentrate on designing a new hybrid index structure for temporal and textual information of Web pages. In particular, we propose to integrate B+-tree, inverted file and a typical temporal index called MAP21-Tree, to handle temporal-textual queries. We study five mechanisms to implement a hybrid index structure for temporal-textual queries, which use different ways to organize the inverted file, B+-tree and MAP-21 tree. After a theoretic analysis on the performance of those five index structures, we conduct experiments on both simulated and real data sets to make performance comparison. The experimental results show that among all the index schemes the first-inverted-file-then-MAP21-tree index structure has the best query performance and thus is an acceptable choice to be the temporal-textual index for future time-aware search engines.

Keywords: Web search, temporal-textual query, temporal information, index structure.

1. Introduction

Web search engines such as Google and Bing have been an important part in people's life. Most people rely on Google to find useful information. The major goal of search engine is to deliver right information to right users quickly, which is generally implemented by a query processing system. In order to achieve this goal, search engines provide many effective ways for users to express their queries precisely, and also develop some efficient algorithms in ranking and indexing. However, previous research on Web search does not pay enough attention to the temporal information in Web pages. For example, it is difficult to express queries like "to find the discount information about Nike in the next week" in Google. On the other side, time is one of essential characteristics of information [1], and most Web pages are related with temporal information, e.g., business news, discount information and so on. Recently, time has been a focus in the area of Web information extraction [2]. Therefore, it is useful and meaningful to utilize temporal information in Web search to enhance traditional search engines, that is, to develop a temporal-textual Web search engine.

In this paper, we focus on the index structures for temporal-textual Web search. Our basic idea is to develop an efficient hybrid index structure to cope with temporal-textual queries, which makes an integration of traditional temporal index and textual index. The most famous textual index is the inverted file structure, so in this paper we use this structure as the basic textual index structure. For temporal index, we adopt the MAP21-Tree [3], which is an efficient temporal index structure in temporal database area. However, there are many choices when integrating inverted file with MAP21-Tree, and in some case we need to introduce B+-Tree as the index structure for update time. Hence, we aims at making a comparison study on those different integration mechanisms, and finally get the best hybrid index structure which has the best performance for temporal-textual queries.

A previous short version of this paper has been published in APWeb 2011 [32]. The major differences between this paper and the previous one are that in this paper we extend the design consideration of temporal-textual Web search engine as well as a time ontology for Web pages. Moreover, we conduct theoretical analysis and further experiments based on a synthetic and a real data set and use more metrics to make performance comparison on the index structures concerned. The main contributions of the paper can be summarized as follows:

(a) We classified the temporal information of Web pages into update time and content time, and introduced the new concept, primary time, into the index process;

(b) We studied and compared five hybrid index structures based on B+-tree, inverted file and MAP21-tree;

(c) We carried out large-scale experiments based on both simulation dataset and real dataset to evaluate the performance of our index structures.

The remainder of this paper is organized as follows. Section 2 introduces the framework of our temporal-textual Web search engine and its main components. Section 3 discusses the five hybrid index structures. Section 4 provides the experimental results. Section 5 describes related work. Finally, we conclude the paper and discuss our future work in Section 6.

2. A Temporal-Textual Web Search Engine

2.1. Time Ontology for Web Pages

Our temporal-textual Web search engine is based on a temporal ontology for Web, which supports different types of Web temporal information (as shown in Fig.1).

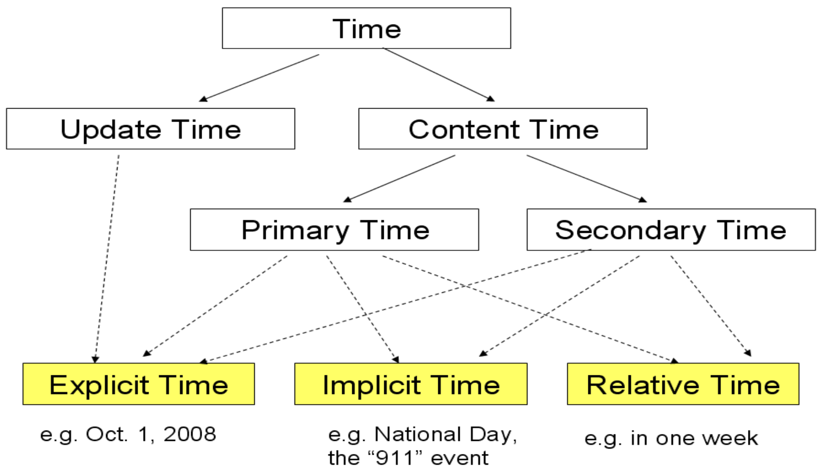

Fig. 1. The Time ontology for Web pages

(a) *Update time*: the update time of a Web page is defined as the crawled date of the Web page.

(b) *Content time*: the content time of a Web page is defined as the temporal information embedded in the main text of the Web page. We use a set of intervals to represent the content time.

(c) *Explicit time*: explicit time can directly be laid in the timeline. Basically, explicit time is a direct entry in the timeline and need not to be transformed.

(d) *Implicit time*: implicit time is a type of fuzzy time which can be mapped as an entry in the timeline with help of some predefined knowledge. Typical implicit time is holiday name or specific event. For example, the “911” event implies an implicit date, which is 2001/09/11.

(e) *Relative time*: relative time is one type of content time of a Web page. It must rely on another time of the Web page to resolve itself as an entry in the timeline. For example, the words “in three days” implies a relative time, since it must rely another date in the text to get the exact date.

(f) *Primary time*: primary time is one type of content time of a Web page. In detail, since there are several content times in one Web page, we will find the most appropriate time that describes the events of the Web page from the set of content times. The most appropriate time is defined as primary time. Generally, when users search in Web through some temporal predicates, they want to get those pages whose primary time of contents is most close to the given temporal information.

(g) *Secondary time*: all the content time of a Web page can be regarded as secondary time, except the primary time.

These different types of time form a relatively complete framework for the representation of temporal information in Web pages. Among those types of time, the update time is always explicit time, since the crawled date of a Web page is definite. The content time can be explicit, implicit or relative. The content time usually contains a set of time period, one of which is chosen as the primary time and the others are secondary time. In our temporal-textual search engine, we only consider the update time and primary time. The reason is that when users search in Web through some temporal predicates, they usually want to get those pages whose primary time is most close to the given temporal information. Otherwise, the temporal predicates in queries will have little influence on the filtering of results, because the contents of most Web pages may contain a lot of and a large range of temporal information.

2.2. The Framework for Temporal-Textual Web Search Engine

Fig.2 shows the system architecture of the temporal-textual search engine. The search engine contains five modules.

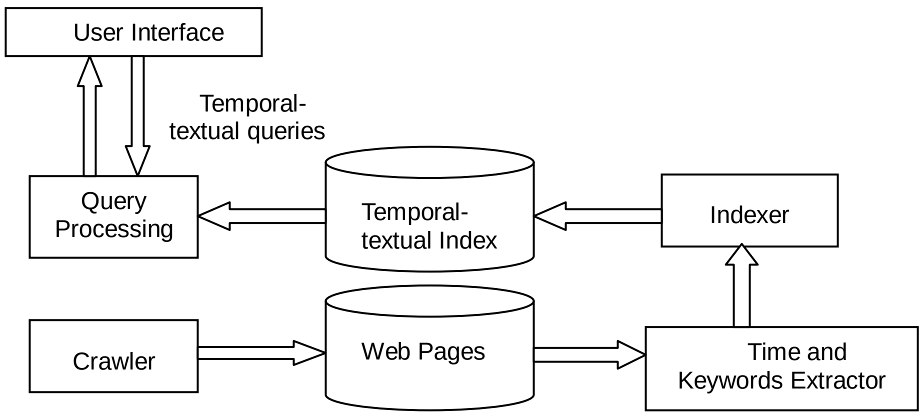

Fig. 2. The system architecture of the temporal-textual search engine

2.2.1 Web Crawler

The Web Crawler periodically crawl the Web to gather Web pages for further information extraction and retrieval. This module is similar with other spiders, except that it stores the crawled pages according to their crawled dates. For example, all the pages obtained in 2008/06/26 will be stored in the directory named “2008-06-26”. In the next extraction step, we will resolve the directory name to get the update time of a Web page.

2.2.2 Time and Keywords Extractor

In this module, the update time, content time, and the keywords of crawled Web pages will be extracted. For update time extraction, this module resolves the directory name and transforms it into standard date format. Since we use the crawled date as directory name, it is easy to get the update time of each Web page. For content time, we use temporal constraints to recognize content time and finally detect the primary time. We first analyze the Web page into a DOM tree. For each leaf node of the DOM tree, we use the TIMEX2 [13] to get the temporal information. After this procedure, we use some temporal constraints to detect the primary time among the whole set of content time. Temporal constraints are rules for primary time. For example, a temporal constraint may be “if a date appears in title, then this date is treated as primary time”. In this module, a set of keywords are also be extracted, which is based on traditional national language processing tools.

2.2.3 Index Constructor

When primary time and keywords are extracted from Web pages, the Index Constructor will create a hybrid temporal-textual index for those crawled Web pages. The hybrid index is based on three basic structures, namely B+-tree, inverted file, and MAP21-tree. We will discuss indexing techniques in detail in Section 4.

2.2.4 Query Processing

The Query Processing module processes user queries and returns ranked results. The main difference between this module and other search engines is that it supports temporal topological queries and uses new rank algorithm. It supports absolute temporal relation and relative temporal relation [30]. For example, it can process such as a user query as “find NBA news after 2008-06-30”, where an absolute temporal relation BEGIN(2008-06-30) is detected and processed. We also design a new ranking algorithm to sort the returned pages. The new ranking algorithm takes into account three factors: temporal information, text similarity, and page importance.

2.2.5 User Interface

The User Interface provides input ways for both temporal queries and textual queries. The textual queries are inputted by a single text box, which is similar with Google. The temporal queries are inputted by a time period box consisting of a start date and an end date. The returned results are shown based on a Timeline [31].

3. Indexer

The indexer is to build hybrid index structures to integrate text keywords and temporal information of Web pages. The inverted file is a standard technique for text indexing, so we adopt it as the basic index structure for text keywords in Web pages. The temporal information contains update time and primary time, in which the update time is regarded as a time instant and the primary time is modeled as a time period. The time granularity is set to day. As the update time is a time instant, we can use B+-tree to organize them or directly put them into inverted files. For the primary time, we adopt the MAP21-tree [3] as the basic index structure. MAP21-tree is designed towards time period and has better performance than other temporal indexes such as R-tree [22].

Table 1. The description of symbols

Symbol	Description
U	The number of update time in the time datasets
P	The number of primary time in the time datasets
K	The number of keywords in the lexicon
$P_U(u)$	The length of the page list of a update time u
$P_P(p)$	The length of the page list of a primary time p
$P_K(k)$	The length of the page list of a keyword k
B_{List}	Storage of page lists
$B_{Tree}(x)$	Storage of a tree of x elements
$T_{I/O}$	The time cost of disk accesses
T_{disk}	The time cost of one disk access
$T_{Tree}(x)$	The time cost to retrieve a tree of x elements
$T_{mg}(x)$	The time cost to merge x elements

We study five hybrid methods: (a) inverted file, B+-tree and MAP-21 triple index, (b) first inverted file then MAP21-tree and B+-tree, (c) first inverted file then MAP21-tree, (d) expanded inverted file, (e) first MAP21-tree then inverted file. In additional, we emphasize the forth hybrid method expanded inverted file. Because the index structure of most related work about temporal text indexing is usually based on inverted file index, expanded inverted file can be considered as the similar method the previous work on building temporal index proposed. We will describe the hybrid index structures and present cost models for each structure. The symbols used in the cost models are listed in Table 1.

3.1. Inverted File, B+-tree and MAP21-tree Triple Index

In the first mechanism, we build indexes separately for keywords, update time and primary time, as shown in Fig.3. The keywords are indexed by an inverted file, which consists of a vocabulary, commonly organized as a B+-tree, and a posting list representing the information about each Web page. The update time is indexed by a B+-tree, while the primary time is organized as a MAP21-tree. Each leaf node of the three trees all points to their corresponding page lists.

A temporal-textual Web search comprises non-temporal keywords and temporal query types. Non-temporal query keywords are retrieved similar to conventional inverted files, temporal query types are passed to the B+-tree for update time and the MAP21-tree for primary time. The final results are produced by merging the page lists from three indexes.

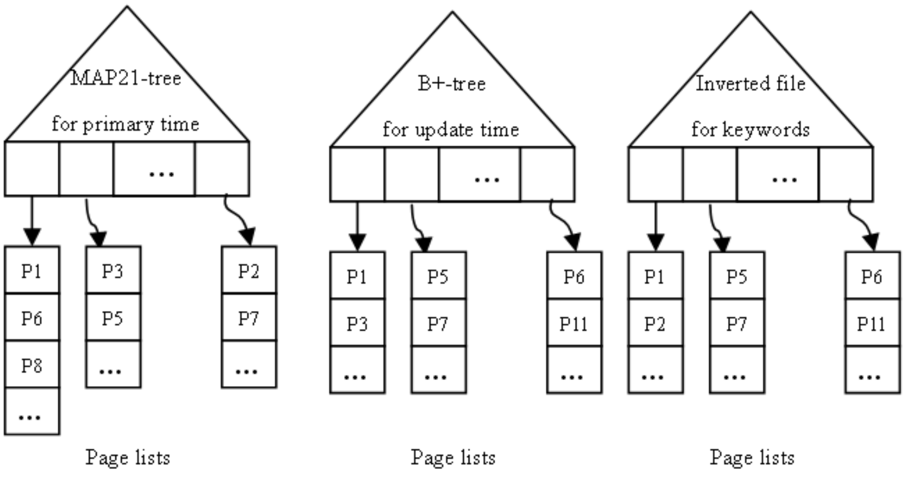

Fig. 3. Illustration of inverted file, B+-tree and MAP21-tree triple Index

The storage in disk comprises the three kinds of page lists and three trees, i.e. $Storage_1 = B^1_{Tree} + B^1_{List}$.

The storage of a tree that has x leaf nodes is $B_{Tree}(x) = O(x)$.

The storage of page lists depends on the length of each list, whose unit is the identifier of a page. Assuming the length of the list whose entry is keyword k is $P_K(k)$, the length of the list whose entry is update time u is $P_U(u)$ and the length of the list whose entry is primary time p is $P_P(p)$, the total length of all lists is $\sum_{u=1}^{U} P_U(u) + \sum_{p=1}^{P} P_P(p) + \sum_{k=1}^{K} P_K(k)$.

Then we have $B_{List}^1 = O\left(\sum_{u=1}^{U} P_U(u) + \sum_{p=1}^{P} P_P(p) + \sum_{k=1}^{K} P_K(k) \right)$.

So, $Storage_1 = B_{Tree}^1 + B_{List}^1$

$$= O(U+P+K) + O\left(\sum_{u=1}^{U} P_U(u) + \sum_{p=1}^{P} P_P(p) + \sum_{k=1}^{K} P_K(k) \right)$$

$$= O\left(\sum_{u=1}^{U} P_U(u) + \sum_{p=1}^{P} P_P(p) + \sum_{k=1}^{K} P_K(k) \right)$$

We can see the main cost of storage in disk is determined by page lists above and the levels of trees. The storage is mainly depended on the total length of all page lists.

Assuming there is a query including m keywords and temporal query types. The online computation includes: (a) the retrieval of the m page lists based on the *m* keywords, (b) the retrieval of the *n* page lists based on the given update time, (c) the retrieval of the *l* page lists based on the given primary time, and (d) the merge of the (*m* + *n* + *l*) page lists to return final results.

The time of loading page lists is determined by the number and total length of page lists. The merge processing is mainly the total length of these page lists.

For the tree that in this structure has U leaf nodes, assuming that the query time is $T_T(U)$.

The merge time for x elements in memory is $T_{mg} = O(x)$.

The time to read a page list containing x bytes is $T_{I/O} = T_{disk} \cdot O(x / B_{section})$.

In the above equation, $B_{section}$ is the page size, which depends on the file system. In our system it is 4K bytes.

Then, $Time_1 = T_{Tree}^1 + T_{I/O}^1 + T_{mg}^1$

$$= T_{Tree}(U+P+K) + \left(\sum_{i=1}^{m} T_{disk} \cdot O\left(P_U(u_i) / B_{section}\right) \right)$$

$$+ \left(\sum_{i=1}^{n} T_{disk} \cdot O\left(P_P(p_i) / B_{section}\right) \right) + \left(\sum_{i=1}^{l} T_{disk} \cdot O\left(P_K(k_i) / B_{section}\right) \right)$$

$$+ O\left(\sum_{i=1}^{m} P_U(u_i) + \sum_{i=1}^{n} P_P(p_i) + \sum_{i=1}^{l} P_K(k_i) \right)$$

For different queries, the query time is determined by three factors. The first factor is the searching time on the three index trees. The second is the merge operations of *m* page lists returned by the keywords query, *n* page lists returned by the update time query, and *l* page lists returned by the primary time query. The third factor is the time to read the (*m* + *n* + *l*) page lists from disk.

3.2. First Inverted File Then MAP21-tree and B+-tree

In this mechanism, two structures are maintained (as shown in Fig.4). The first one is an inverted file, each of whose leaf nodes points to a MAP21-tree which contains the primary time in the posting lists corresponding to the leaf node. The second one is a B+-tree used for indexing update time.

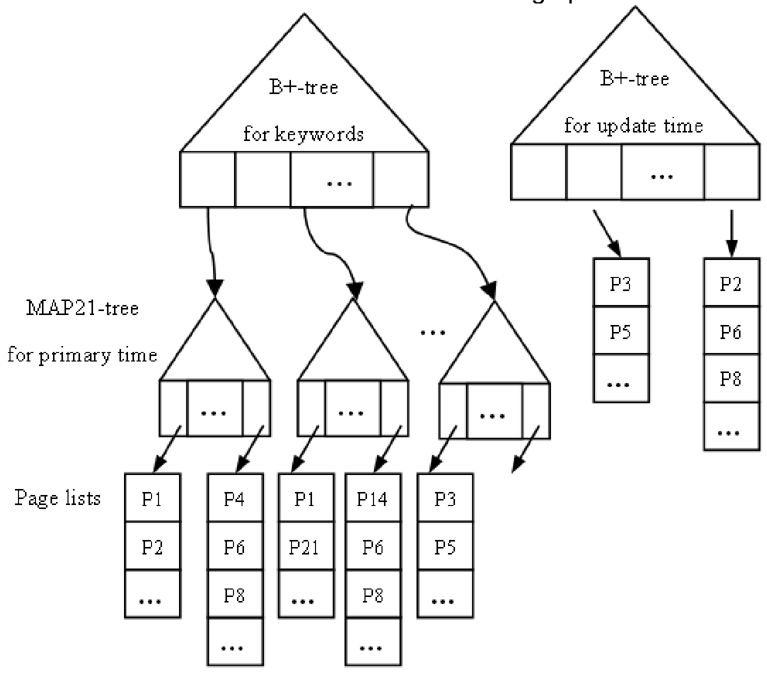

Fig. 4. Illustration of first inverted file then MAP21-tree and B+-tree index structure

Assuming $P_T(t)$ is the length of a page list whose entry is a ptime-keyword *p*.

The storage in disk includes these page lists, MAP21-trees pointed by K keywords and one B+-tree besides the page list whose entry is update time and one B+-tree. So,

$$\begin{aligned} Storage_2 &= B_{Tree}^2 + B_{List}^2 \\ &= K \cdot O(P) + O(U) + O(K) + O\left(\sum_{t=1}^{T} P_T(t) + \sum_{u=1}^{U} P_U(u) \right) \end{aligned}$$

In fact, a MAP21-tree in this structure may not index all P leaves as in the first structure, so the scale of MAP21-trees is smaller. Thus we can see the cost of storage in disk is mainly caused by the total length of page lists whose entry is ptime-keyword and the length of page list whose entry is update time.

If we input *m* keywords, one primary time period and one update time instant, the online computation has three parts: (a) finding the *m* leaf nodes in the invert file according to the *m* query keywords, and then searching the corresponding MAP21-subtrees pointed by the *m* leaf nodes, and then loading the corresponding page lists from disk; (b) searching the B+-tree for the update time, and loading the corresponding page lists from disk; (c) merging the page lists returned by MAP21-tree and those returned by the update time B+-tree to generate final results. So,

$$\begin{aligned} Time_2 &= T_{Tree}^2 + T_{I/O}^2 + T_{mg}^2 \\ &= m \cdot T_{Tree}\left(\overline{M}\right) + T_{Tree}(u) + \left(\sum_{i=1}^{m} T_{disk} \cdot O\left(P_T\left(u_i\right) / B_{\text{section}} \right) \right) \\ &+ \left(\sum_{i=1}^{n} T_{disk} \cdot O\left(P_U\left(u_i\right) / B_{\text{section}} \right) \right) + O\left(\sum_{i=1}^{m} P_T(t_i) + \sum_{i=1}^{n} P_U(u_i) \right) \end{aligned}$$

In the above equation, $\overline{M}$ is the average number of leaf nodes in all the *m* MAP21-trees.

There are three factors that affect the query time. The first is the retrieval of *m* MAP21-subtrees. The second is the merge operations of *m* page lists returned by *m* keywords. The third factor is the time to read the (*m* + *n*) page lists from disk, supposing the searching on the update time B+-tree returns *n* page lists.

3.3. First Inverted File Then MAP21-tree

Inverted file is built by B+-tree, the leaf node of B+-tree points to the inverted list of the keyword. In the structure, the leaf node points to a MAP21-tree which is build on time information of the inverted list of the corresponding keyword, then the leaf node of the MAP21-tree points to a set of page lists whose entry is depended on a pair of a keyword and a primary time. So n leaf nodes of B+-tree points totally to not n page lists but n MAP21-subtree. A pair of a keyword and a primary time period is named primary-keyword. We can know every MAP21-subtree in the structure may not index all primary times, because each word doesn't exist in every Web page, then it doesn't include all primary times, the MAP21-subtree is smaller than one unique MAP21-tree and the size of page lists pointed by leaf node of MAP21-subtree is almost smaller than MAP21-tree's. To update time, we don't index the type of time information using B+-tree like the structure above, it is directly inserted the detail of a page list of MAP21-tree, which is the difference from the above index structure. We show the structure in Fig.5.

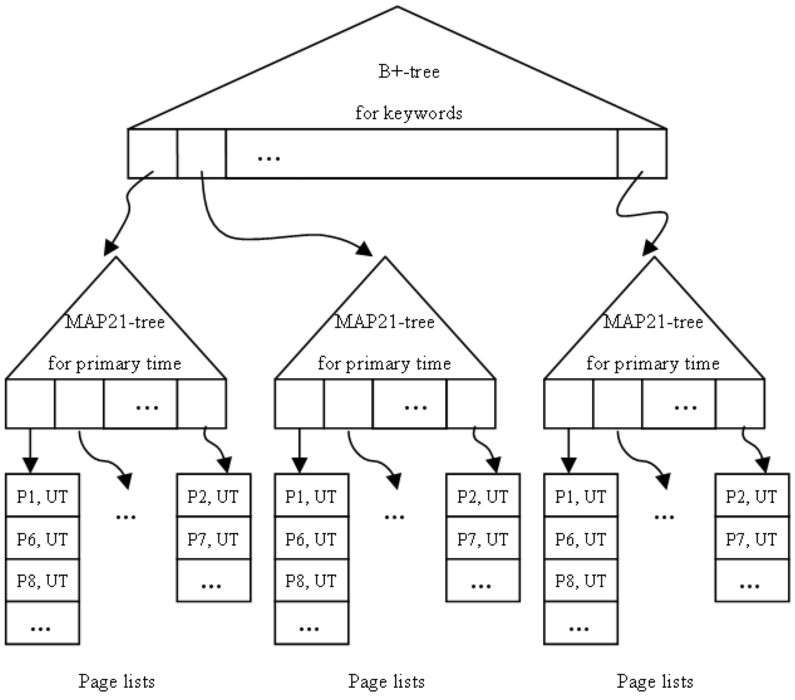

Fig. 5. The illustration of first inverted file then MAP21-tree(UT = Update Time)

The storage in disk includes these page lists, MAP21-trees pointed by K keywords and one B+-tree. So,

$$Storage_3 = B^3_{Tree} + B^3_{List} = K \cdot O(P) + O(U) + O\left(\sum_{t=1}^{T} P_T(t)\right)$$

The main cost of storage in disk is caused by the total length of page list whose entry is a putime-keyword.

If we input m keywords, one primary time period and a update time instant, the online computation has three parts: (a) to retrieve the m query keywords and then search the corresponding MAP21-subtrees by these leaf nodes of m keywords, and the loading of corresponding page lists m from the disk; (b) to retrieve the map value of the update time instant from corresponding page lists n which is in memory from disk; (c) to merge the n page lists. So,

$$Time_3 = T^3_{Tree} + T^3_{I/O} + T^3_{mg}$$
$$= m \cdot T_{Tree}\left(\overline{M}\right) + \left(\sum_{i=1}^{m} T_{disk} \cdot O\left(P_U\left(u_i\right) / B_{section}\right)\right) + O\left(\sum_{i=1}^{m} P_T\left(t_i\right)\right)$$

There are three factors that effect the query time. One factor is the retrieval of m MAP21-subtrees, the second factor is the merge operations of n page lists whose entry is the keyword and its corresponding MAP21-tree. The third factor is the time to read the m pages lists from disk. The pages in a page list whose entry is a primary-keyword is a subset of pages in the page list whose entry is the corresponding keyword of primary time period, so the length of a page list whose entry is a primary-keyword is reduced smaller than the above structure.

3.4. Expanded Inverted File

Inverted file is built by B+-tree, the leaf node of B+-tree points to the inverted list of the keyword. Because the index structure of most related work about temporal text indexing is usually based on inverted file index. The basic lookup operation in text-indexing is to retrieve the document identifiers of all document s that contains a particular word w. we consider designing the similar structure. The difference is the details of the page list pointed by the leaf node of B+-tree for inverted file. In the page list, there is not only corresponding set of URLs of Web pages but also the corresponding update time and the primary time period of one Web page. As shown in Fig.6.

The storage in disk includes the page lists and one B+-tree.

So, $Storage_4 = B^4_{Tree} + B^4_{List} = O(K) + O\left(\sum_{t=1}^{T} P_T(t)\right)$

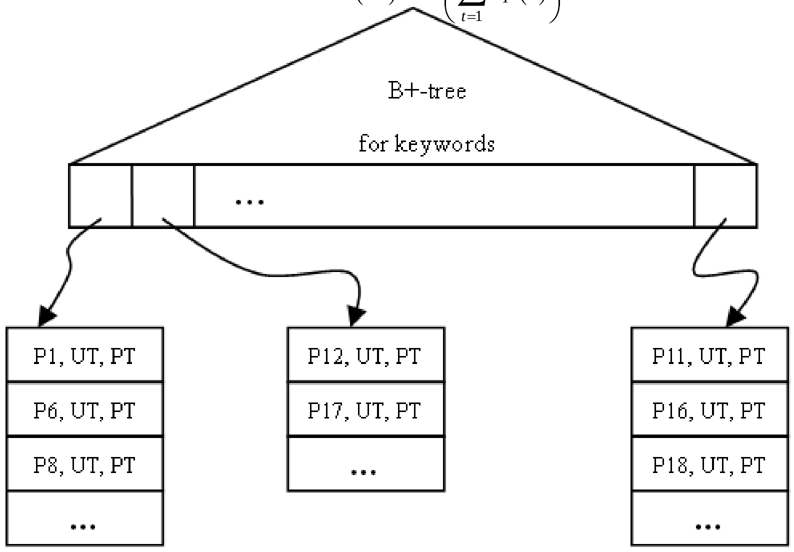

Fig. 6. Illustration of expanded inverted file structure(UT = Update Time, PT = Primary Time)

The main cost of storage in disk is caused by the total length of page lists whose entry is a keyword.

Assume there is the input of *m* keywords, one update time instant and a primary time period. The online computation includes: (1) the retrieval of the m keywords and the loading of corresponding page lists from disk; (2) the merge of the m page lists filtrated the time period not included in and error time instant in memory. So,

$$Time_4 = T_{Tree}^4 + T_{I/O}^4 + T_{mg}^4$$
$$= T_{Tree}(K) + \left(\sum_{i=1}^{m} T_{disk} \cdot O\left(P_T\left(t_i\right) / B_{\text{section}}\right)\right) + O\left(\sum_{i=1}^{m} P_T\left(t_i\right)\right)$$

There are also main factors for the query time. One factor is the merge of m page lists, at the same time, to filtrate the Web pages included wrong time information that users do not want. The other factor is the time to read the page lists from disk.

3.5. First MAP21-tree Then Inverted File

As shown in Fig.7, a MAP21-tree is built on all the primary time periods of Web pages. The leaf node of the tree points to a B+-tree containing the keywords that are included in the corresponding time periods. The leaf node of every B+-tree points to a page list. Each page list contains the update time of Web pages and the corresponding set of URLs.

The main storage in disk includes the page lists, B+-trees pointed by P primary times and one MAP21-tree. So,

$$Storage_5 = B_{Tree}^5 + B_{List}^5$$
$$= P \cdot O(K) + O(P) + O\left(\sum_{t=1}^{T} P_T(t)\right)$$

The main cost of storage in disk is caused by the total length of page lists whose entry is a putime-keyword.

If we input *m* keywords, one primary time period and a update time instant, the online computation has three parts: (a) searching the MAP21-tree and getting *n* primary time periods; (b) for each time period, retrieving the *m* keywords through the invert file, and the loading the corresponding page lists; (c) merging the *m* page lists. So,

$$Time_5 = T_{Tree}^5 + T_{I/O}^5 + T_{mg}^5$$
$$= m \cdot T_{Tree}\left(\overline{K}\right) + T_{Tree}(P) + \left(\sum_{i=1}^{m} T_{disk} \cdot O\left(P_T\left(t_i\right) / B_{\text{section}}\right)\right) + O\left(\sum_{i=1}^{m} P_T\left(t_i\right)\right)$$

In the above equation, $\overline{K}$ is the average number of leaf nodes in the B+-trees.

There are three factors that affect the query time. One factor is the retrieval of *m* B+-subtree, the second factor is the merge operations of page lists. The third factor is the time to read the *n* pages lists from disk.

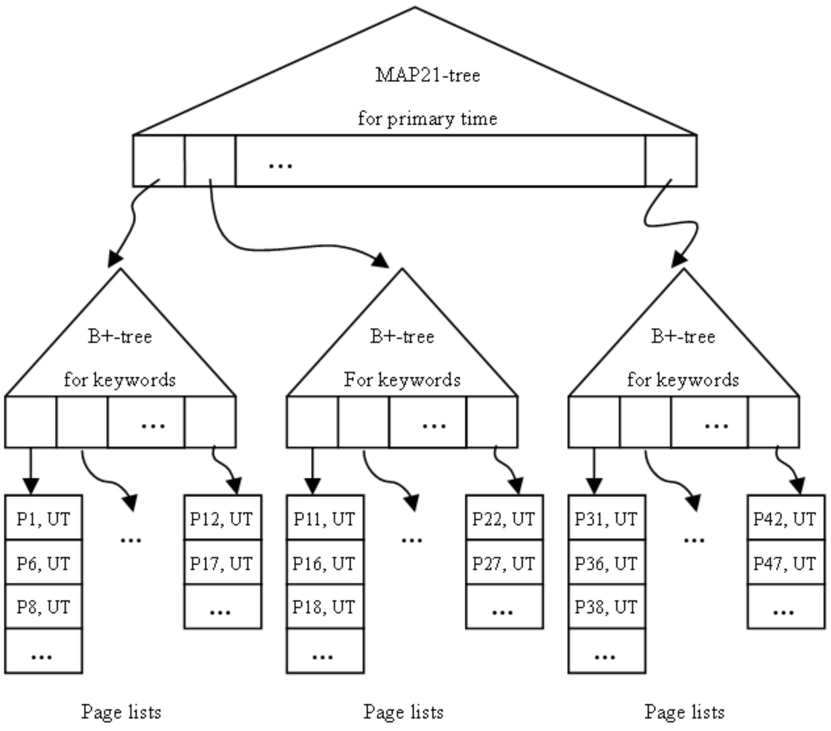

Fig. 7. Illustration of first MAP21-tree then inverted file (UT = Update Time)

There are some major differences between this structure and the index structure in Section 3.3, which refers to first inverted file then MAP21-tree. In the case that the index structure is first inverted file then MAP21-tree, if we input one keyword, one update time and one primary time period, we will first find the leaf node of keyword from one B+-tree, then retrieve one corresponding MAP21-tree to get right page lists from disk. If the index structure is first MAP21-tree then inverted file, the first step is finding *m* leaf nodes of primary time from one MAP21-tree according to the given primary time period, then retrieve each corresponding B+-tee to get right page lists from disk. Hence, the main difference between these two structures is the number of B+-subtree that is read from disk. The number of B+-subtrees will increase with the increasing of the range of primary time query.

4. Experiments

In this section, we will implemented the five hybrid index structures and make comparison experiment to show the different performance of those index structures under different workloads. For simplicity, we use the following notation to represent the five hybrid index structures discussed in the previous section:

Structure 1: inverted file, B+-tree and MAP-21 triple index.
Structure 2: first inverted file then MAP21-tree and B+-tree.
Structure 3: first inverted file then MAP21-tree.
Structure 4: expanded inverted file.
Structure 5: first MAP21-tree then inverted file.

We mainly evaluate the query performance of the five index structures. In order to get a comprehensive result, we use five types of queries in the experiment. Those queries are:

TYPE 1: keyword query
TYPE 2: keywords + update time instant
TYPE 3: keywords + primary time instant
TYPE 4: keywords + update time instant + primary time instant
TYPE 5: keywords + update time period + primary time period

In our experiments, we focus on the index size, I/O number, and run time of each index structure under given workload and temporal-textual queries. Run time contains I/O number and memory time, and I/O time costs much longer than memory time. So we not only compare the run time but also compare the I/O number.

We used two types of dataset in our experiment: a simulated dataset and a real dataset. A simulated dataset is more regular than a real dataset, so the experiment on simulated dataset may achieve the desired result. A real dataset is more complex, so we experiment on it beside the simulated dataset. In the following part, we will describe the experimental results on simulation and real dataset respectively.

4.1. Simulation Experiment

4.1.1 Settings and Dataset

In our simulation experiment, we manually generate a simulated dataset. In this dataset, each Web page has one update time instant, one primary time period, five different keywords. All the time included in the dataset is limited in one year, i.e., the update time is set as a day in 2009, and the primary time is set as a time period in 2009. For simplicity, we use a number ranging from 1 to 100000 to represent a keyword. As a result, a generated record representing a Web page is as follows:

<URL, UT, PTs, PTe, key1, key2, key3, key4, key5>

Here, *UT* represents the update time, *PTs* and *PTe* represent the start and end date of primary time.

In our experiment, we generate three simulated datasets. They contain 1095 thousand records, 1825 thousand records, and 2555 thousand records, respectively. In the following text, we use T1095, T1825, and T2555 to indicate the three types of simulated datasets. We want to know whether the result may change with the incremental datasets.

We run our experiment in a computer with an Intel Core 2.00 GHz CPU, 2 GB RAM, using Microsoft Window 7.

4.1.2 Comparison of Five Hybrid Index Structures

First, we compare the index size (Mbytes) of five hybrid index structures. We generate three trace, we show them respectively in Table 2, Table 3 and Table 4.

Table 2, Table 3 and Table 4 show that Structure 1 has the biggest total index size. The size of Structure 2 is bigger than that of Structure 4 but smaller than that of Structure 1, and Structure 3 and Structure 5 has smaller size than Structure 4. So the index size of Structure 3 and Structure 5 is the smallest in the five index structures.

Table 2. Index size for five hybrid index structures under the T1095 dataset (MBytes)

	Keyword index	Update time index	Primary time index	Total index size
Structure 1	684.62	136.86	136.86	958.34
Structure 2	0	136.86	700.96	837.82
Structure 3	0	0	759.78	759.78
Structure 4	802.25	0	0	802.25
Structure 5	759.81	0	0	759.81

Table 3. Index size for five hybrid index structures under the T1825 dataset (MBytes)

	Keyword index	Update time index	Primary time index	Total index size
Structure 1	1164.86	228.13	228.13	1612.12
Structure 2	0	228.13	1191.67	1419.80
Structure 3	0	0	1289.69	1289.69
Structure 4	1360.90	0	0	1360.90
Structure 5	1289.74	0	0	1289.74

Table 4. Index size for five hybrid index structures under the T2555 dataset (MBytes)

	Keyword index	Update time index	Primary time index	Total index size
Structure 1	1621.13	319.38	319.37	2259.88
Structure 2	0	319.37	1657.89	1977.26
Structure 3	0	0	1795.14	1795.14
Structure 4	1895.58	0	0	1895.58
Structure 5	1795.21	0	0	1795.21

Second, we compare page I/O numbers of the five index structures on the basis of the five types of queries mentioned before. The page size is set as 4 Kbytes. The total I/O number of every query includes the retrieval of tree nodes and page lists from disk. We generate 300 queries randomly, each of which contains three keywords and one update time and one primary time, and calculate the average I/O number of the 300 queries. The results are shown in Table 5, Table 6 and Table 7. Additionally, the reason why we choose three words is that a users' query usually contains one to three keywords in the Web search environment.

Table 5. Page I/O# for five hybrid index structures under the T1095 dataset

	Type 1	**Type 2**	**Type 3**	**Type 4**	**Type 5**
Structure 1	1687	1779	1719	1811	2052
Structure 2	1690	1782	40	131	366
Structure 3	1695	1695	40	40	95
Structure 4	1701	1701	1701	1701	1701
Structure 5	8252	8252	37	37	107

Table 6. Page I/O# for five hybrid index structures under the T1825 dataset

	Type 1	**Type 2**	**Type 3**	**Type 4**	**Type 5**
Structure 1	2550	2707	2604	2762	3185
Structure 2	2554	2712	59	218	629
Structure 3	2561	2561	59	59	159
Structure 4	2571	2571	2571	2571	2571
Structure 5	9105	9105	56	56	171

Table 7. Page I/O# for five hybrid index structures under the T2555 dataset

	Type 1	**Type 2**	**Type 3**	**Type 4**	**Type 5**
Structure 1	3467	3687	3542	3762	4351
Structure 2	3473	3694	79	300	873
Structure 3	3482	3482	79	79	217
Structure 4	3495	3495	3495	3495	3495
Structure 5	10126	10126	74	74	227

In Table 5, Table 6 and Table 7, we can see the number of blocks read from disk based on Structure 1, 2, 3 and 4 is similar for Type 1. The number of blocks based on Structure 3 and Structure 4 is smaller than the other three structures for Type 2, and the two structures have almost no difference or little difference. For Type 3, Structure 2, Structure 3 and Structure 5 have obvious advantages over the other two structures. Structure 3 and Structure 5 have good performance for Type 4 when reading blocks from disk. Structure 3 and

Structure 5 are better than the other structures for Type 5. In addition, Structure 3 is little better than Structure 5 as the increased number of Web pages.

Table 5 to Table 7 show that Structure 3 has the smallest I/O cost for five types of queries, Structure 5 is better for three types of queries except Type 1 and Type 2. Structure 3 and Structure 5 both have some subtrees. The advantage of subtree is that it filters some unmatched Web pages according to the given keywords or primary time.

Third, we compare the run time of five structures for five types of queries. Run time is calculated from the input of query to the output of the right URL set. The queries are the same above. The results are shown in Table 8, Table 9 and Table 10. The unit of time is second.

Table 8. Run time for five hybrid index structures under the T1095 dataset (seconds)

	Type 1	Type 2	Type 3	Type 4	Type 5
Structure 1	2.15	2.35	2.34	2.37	2.36
Structure 2	2.21	2.48	0.06	0.17	0.45
Structure 3	2.33	1.29	0.07	0.07	0.16
Structure 4	2.24	1.26	1.38	1.37	1.66
Structure 5	9.31	8.28	0.06	0.06	0.18

Table 9. Run time for five hybrid index structures under the T1825 dataset (seconds)

	Type 1	Type 2	Type 3	Type 4	Type 5
Structure 1	3.10	3.18	3.17	3.21	3.26
Structure 2	3.15	3.27	0.08	0.25	0.71
Structure 3	3.33	2.05	0.10	0.09	0.25
Structure 4	3.16	1.95	2.08	2.09	2.17
Structure 5	10.30	9.19	0.09	0.09	0.24

Table 10. Run time for five hybrid index structures under the T2555 dataset (seconds)

	Type 1	Type 2	Type 3	Type 4	Type 5
Structure 1	4.05	4.23	4.21	4.22	4.30
Structure 2	4.48	4.62	0.13	0.44	1.30
Structure 3	4.22	3.17	0.14	0.14	0.39
Structure 4	3.52	2.63	2.80	2.80	2.93
Structure 5	11.21	9.90	0.11	0.11	0.34

Run time includes three main parts. They are the time for retrieving corresponding trees, the time for reading page lists from disk and the time for merging page lists. The run time in the first and the second part is mainly determined by the page I/O numbers listed in Table 5 to Table 7. The time for merging page lists is much smaller than the time for reading page lists from disk. So we can get the similar comparison result as shown in Table 5 to Table 7. We can see the run time based on Structure 1, 2, 3 and 4 is similar for Type 1. Structure 3 and Structure 4 is faster than the other three structures for Type 2. In addition, these two structures have almost no difference or little difference in run time for Type 2. Structure 2, Structure 3 and Structure 5 have obvious advantages over the other two structures for Type 3. Structure 3 and Structure 5 have good performance for Type 4. Structure 3 and Structure 5 are better than the other structures for Type 5. Additionally, Structure 3 is little better than Structure 5 as the increased number of Web pages. Generally, Structure 3 wins the best in the measurement of run time.

4.1.3 Rebuilt Time of Five Hybrid Index Structures

We consider the update cost now. When Web pages update are small, update pages can be inserted into the specified location directly. Once the Web pages update are large, rebuild index is a good choice. We compare rebuild cost of five hybrid index structures here. The result of three different simulated datasets is shown in Table 11.

Table 11. Rebuild time for five hybrid index structures under the T1095, T1825 and T2555 dataset (seconds)

	T1095	T1825	T2555
Structure 1	1995.18	7249.17	11465.50
Structure 2	4230.64	8682.05	12781.80
Structure 3	4877.58	8641.93	14312.10
Structure 4	3891.86	8180.55	12537.10
Structure 5	4097.65	7713.05	10610.20

Table 11 shows that the rebuild time of Structure 1 costs least under the three simulated datasets. Structure 4 and Structure 5 cost longer than Structure 1, and the rebuild time of Structure 4 and Structure 5 has little difference. Structure 2 and Structure 3 cost longest. Although the rebuild time of Structure 3 cost longest, its query performance is very dominant, so we can ignore the rebuild cost.

4.2. Experiment on real dataset

4.2.1. Settings and Dataset

We choose the real dataset from the corpus of SouGou lab (http://www.sogou.com/labs/) which records games, sports, IT, domestic and international news in May 2008 from some news sites.

The experiment on real dataset is more complicated than the simulation experiment. In this experiment, we simply describe how to exact the update time and the primary time in one real Web page. We use the real dataset from news Web pages, and the news pages have their own characteristics. In the news, publish time is usually as the update time and primary time is often appears in the first paragraph. The exaction of primary time is more complicated than update time. If there is one time instant in the first paragraph, we consider it as the primary time; If there is two or more time instants in the paragraph, we consider choosing the nearest instant to update time instant as primary time; If there are time instants and time periods, we first choose the time period as the primary time of the Web page.

Keywords of every Web page are exacted by a tool called ICTCLAS (http://ictclas.org/), which is the most efficient tool for the Chinese words segmentation. Each word is mapped a value in memory by ELFHASH function.

We use 250 thousand news Web pages as our real dataset and extract approximately 210 thousand different keywords. The experimental computer environment is the same as that in the simulation experiment.

4.2.2. Comparison of Five Hybrid Index Structures

We still use the five query types to measure the index size. Page I/O#, and run time of each hybrid index structure.

First, we compare the index size (MBytes) of five hybrid index structures. The results are shown in Table 12.

Table 12. Index size for five hybrid index structures under the real dataset (MBytes)

	Keyword index	Update time index	Primary time index	Total index size
Structure 1	1307.69	31.25	31.25	1370.19
Structure 2	0	31.25	1315.04	1346.29
Structure 3	0	0	1425.14	1425.14
Structure 4	1528.07	0	0	1528.07
Structure 5	1425.22	0	0	1425.22

Table 12 shows that Structure 1 has the largest size, and Structure 2 is worse than Structure 4 but better than Structure 1. Besides, the index size of Structure 3 and Structure 5 is the smallest in the five index structures. Therefore, we have the same conclusion as what we got in the simulation experiment.

Second, we compare page I/O# of five index structures for five query types. The page size is set as 4 KBytes. We choose 300 queries of three keywords from the search log, one update time and one primary time randomly and calculate the average of the 300 queries. We show them in Table 13.

Table 13. Page I/O# for five hybrid index structures under the real dataset

	Type 1	Type 2	Type 3	Type 4	Type 5
Structure 1	481	745	738	1001	2950
Structure 2	485	749	27	291	1780
Structure 3	486	486	27	27	67
Structure 4	485	485	485	485	485
Structure 5	3788	3788	26	26	79

Table 13 shows that the number of blocks read from disk based on Structure 1, 2, 3 and 4 is similar for Type 1. The number of blocks based on Structure 3 and Structure 4 is smaller than those of the other three structures for Type 2 and the two structures have almost no difference or little difference. For Type 3, Structure 2, Structure 3 and Structure 5 have obvious advantages over the other two structures. Structure 3 and Structure 5 have good performance for Type 4 when reading blocks from disk. Structure 3 and Structure 5 are better than the other structures for Type 5, in addition, Structure 3 is little better than Structure 5 as the increased number of Web pages.

As a result, Structure 3 is the best index structure for five types of query, Structure 5 is better for three types of queries except Type 1 and Type 2. This is the same as what we get in the simulation experiment.

Third, we compare the run time of five structures for five types of search. Run time is the process which is from the input of query user need to the output of the right URL set. The queries are the same above. We show them in Table 14. The unit of time is second.

Table 14. Run time for five hybrid index structures under the real dataset (seconds)

	Type 1	Type 2	Type 3	Type 4	Type 5
Structure 1	0.96	1.30	1.31	1.62	3.85
Structure 2	1.03	1.34	0.08	0.41	2.14
Structure 3	1.12	0.53	0.08	0.08	0.14
Structure 4	1.11	0.54	0.60	0.58	0.59
Structure 5	6.00	5.30	0.08	0.08	0.15

4.2.3. Rebuilt Cost of Five Hybrid Index Structures

We consider the update cost now. When Web pages update are small, update pages can be inserted into the specified location directly. Once the Web pages update are large, rebuild index is a good choice. We compare rebuild cost of five hybrid index structures under the real dataset. The result is shown in Table 15.

Table 15. Rebuild time for five hybrid index structures under the real dataset (seconds)

	Structure 1	Structure 2	Structure 3	Structure 4	Structure 5
Rebuilt time	9146.88	29235.10	31310.14	14170.90	3829.13

Table 15 shows that the rebuild time of Structure 5 costs least, and Structure 3 has the longest rebuild time. Although the rebuild cost of Structure 3 is larger, compare with the query performance, the rebuild cost is affordable.

5. Related Work

5.1. Temporal Information Extraction and Retrieval

Traditional commercial search engines, such as Google, Bing, and Baidu, have noticed the value of temporal information in Web search. They all provide some ways for users to perform a Web search based on time. E.g. Google uses the daterange option to express a temporal predicate. However, those commercial search engines only support the crawled dates of Web pages, i.e., users can only query Web pages towards their creation dates in database. There are also some other temporal information retrieval systems which use similar methods as Google to process temporal information of Web pages, such as Goo [4], Infoseek [5], Namazo [6], Chronica [7], and so on. Generally, there is a gap between the crawled time and content time of a Web page. For instance, if a news page reports that in Aug 8th, 2008 the Olympic Games will be held in Beijing, China, but it is posted and crawled in Jun. 21st, 2006. In such case, "Jun. 21st, 2006" will be regarded as the temporal information of this news page, but unfortunately it does not report the right temporal information of the page. To our knowledge, there are few search systems considering the temporal information embedded in Web pages [1, 7]. The system presented in [8] aims at extracting and indexing the content time of Web pages, but it only considered the business hour extraction, and can not deal with implicit time such as "today", "Tuesday", "Last Christmas", and so on. Other work in Natural Language Processing (NLP) focused on temporal information extraction and annotation from text. There are a lot of tools which can extract temporal elements from text. Many of them are towards English text, such as Lingua::EN::Tagger [9] and TempEX [10]. In recent years, some temporal extraction tools for non-English languages were also presented. For example, KTX was a temporal information extracting tool for Korean text [11], and CTEMP was for Chinese text [12]. Most of temporal information extraction tool in NLP are based on the temporal information annotation standard TIMEX2 [13] or TimeML [14]. Though temporal information extraction in NLP is relative mature, little effort has been done to combine them into Web search engines.

5.2. Temporal-Textual Indexes

The current temporal text indexing is mainly to the versioned document collections such as Web archives [15, 16]. There have been some indexing approaches on directly addressing the issue of temporal-textual indexing. Anick and Flynn [17] have pioneered this research to support versioning in a full-text index on bitmaps for terms in current versions, and delta change records to track incremental changes to the index backward over time. The disadvantage is the costly recreation of previous states. Recent work in [19, 23-26] and their earlier proposals concentrate on the problem of supporting text-containment queries and neglect the relevance scoring of results. Stack [20] reports practical experiences made when adapting the open source search engine Nutch to search Web archives. Weikum et al. address the temporal dimensions completely by extending the inverted files index to make it ready for temporal search and implement the time-travel text search in the FluxCapacitor prototype [27, 28]. In contrast, research in temporal databases has produced several index structures tailored for time-evolving databases. A comprehensive overview of the state-of-art is available in [29]. Unlike the inverted file index, their applicability to text search is not well understood.

Temporal indexes have been deeply studied in temporal database area. In temporal database, two dimensions of time, which are valid time and transaction time, may be considered in the index [21]. Therefore, R-tree [22] and its variation as the access structures for spatial data may also be used as a temporal index. Among all the temporal indexes proposed before, the MAP21-tree [3], which utilizes standard B+-trees, provides efficient indexing of valid time period, and has good performance in time instant query and range query. The idea of MAP21-tree is to map a two-dimensional time period to a one-dimensional number and then to use a B+-tree to build the index structure. So in this paper we choose the MAP21-tree as the basic temporal index structure.

The index structure of most related work about temporal-textual indexing is usually based on inverted file index [18]. However, the main difference between our work and previous researches is that we consider to index both update time and content time for Web pages, while previous temporal-textual indexes usually focused on indexing update time, because they are designed for Web archive system or document versioning.

In addition, in this paper we introduce the concept of primary time. Hence, the temporal-textual index structures studied in this paper involve keywords, update time, and content time. Traditional models in information retrieval have been widely studied since 1970s, among which the Vector Space Model (VSM) [15, 16] and the Probabilistic Relevance model (PPR) [16, 17, 20] are two representatives. In VSM, all the keywords are represented into high dimension feature vectors rather than representing the keywords with binary value. The problem of VSM is it considers little about the relationship between keywords. The BM25 model, as a popular Probabilistic Relevance model [17], ranks Web pages based on their probability of relevance with the query. This model needs to know the information about which Web pages are relevant with the query, which is very difficult to realize in a large dataset.

Pagerank ranking model [21] is an offline ranking algorithm which is based on the number of Web pages that are linked by other pages in the whole Web, and the quality of the sources of the links. The problem of Pagerank is it only considers links but ignores the similarity between the query and Web pages.

Most of time-related Web search now concentrates on Web archive system [22-24]. A Web archive system is used to store and manage historical Web pages and then provides evolutional information of the Web. The history of a Web page is typically captured by the versioning technique, i.e., the new version of a Web page is stored with an explicit update timestamp. However, Web archive systems only consider the update timestamps of Web pages. They do not take into account the content time of Web pages, which is much different from the research scope of this paper.

In recent years, several researchers have studied ways to find fresh Web pages. The TimedPageRank algorithm [25] was proposed in a Web-based literature searching prototype. It uses the posted time of paper to perform the ranking process. If we map it into a general Web search engine, the posted time of paper can be regarded as the publication time or update time of Web page. It can not support queries focusing on the content time. In [26], a temporal search system for business hours was studied, which tried to answer such questions 'Which shops are open and in which time are they open'. In this system, the time granularity was restricted in hour, e.g., '9:00 AM'. Besides, it does not support implicit time, such as Christmas, the National Day. So it is not suitable for general Web search engines.

The language modeling approach for information retrieval was first in 1998 [5]. Its basic idea is to estimates the probability of the query given the language model of a Web page, and ranks Web pages according to those probabilities. There are also some variants of this approach [27, 28, 29]. Previous studies have shown that the language model has a better performance than traditional models such as VSM and BM25, and the experimental results in this paper also proved this truth.

There are also some recent works focusing on temporal language models [18, 19, 30], which integrate temporal information into the framework of language models. In the literature [19], a time-based language model was proposed, which emphasized that recent documents could be better to satisfy users' needs. However, this model only concentrates on the publication time of Web pages, and therefore is useful for querying recent events but is not a general framework. A similar work could be found in [30], which also focused on the publication time. The recent work in [18] integrated the content time of Web pages into the language model. In this approach, the researchers proposed the assumptions of temporal relevance, which first had a filter process to filter all the Web pages containing no temporal references in their contents, and then used the triangle distribution model to simulate the probabilities of the querying time appearing in each Web page. However, this model does not consider the relationship between the keywords and the temporal references in the page content.

6. Conclusions

In this work we have designed and implemented five hybrid index structures for temporal-textual Web search and studied the performance of these index structures. We conduct a comprehensive experiment on both the simulated and real datasets, and use five temporal-textual query types to evaluate the index size, page I/O#, and run time of each hybrid index structure. Both the simulation and real experimental results show that the index structure “first inverted file then MAP21-tree” has the best performance among the five index structures. Therefore, it should be an acceptable choice for indexing temporal and text information in a temporal-textual Web search engine.

In the future research, we will focus on the update performance of the index structure, and integrate the hash policy to improve the update and search performance. Another work will be the compression of the index, since the index size of the hybrid index structure is still too big for Web search.

Acknowledgements. This work is supported by the National Science Foundation of China (no. 70803001), the Open Projects Program of National Laboratory of Pattern Recognition (20090029), the Key Laboratory of Advanced Information Science and Network Technology of Beijing (xdxx1005), and the USTC Youth Innovation Foundation.

References

1. Alonso, O., Gertz, M., Yates, R., On the value of temporal information in information retrieval, In Proc. Of SIGIR'07, 35-41. (2007)
2. Jensen, C., Temporal Database Management, PhD Thesis, http://www.cs.auc.dk/~csj/Thesis/. (2000)
3. Nascimento, M., Dunham, M., Indexing Valid Time Databases via B+-Trees. IEEE Transactions on Knowledge and Engineering. Vol.11(6), 929-947. (1999)
4. Goo, http://www.goo.ne.jp/
5. Infoseek, http://www.infoseek.co.jp/
6. Namazu, http://www.namazu.org/
7. Deniz, E., Chris, F., Terence, J., Chronica: a temporal Web search engine. In Proc. Of ICWE'06, 119-120. (2006)
8. Taro, T., Katsumi, T., Temporal and Spatial Attribute Extraction from Web Documents and Time-Specific Regional Web Search System, in Proc. Of W2GIS'04, 14-25. (2004)
9. Lingua::EN::Tagger, http://search.cpan.org/~acoburn/ Lingua-EN-Tagger
10. Mani, I., Wilson, G., Robust temporal processing of news. In Proc. of ACL’00, 69-76. (2000)

11. Jang, S., Baldwin, J., Mani, I., Automatic TIMEX2 tagging of Korean news, ACM Trans. Asian Lang. Inf. Process, Vol.3(1), 51-65. (2004)
12. Wu, M., Li, W., Lu, Q., Li, B., CTEMP: A Chinese Temporal Parser for Extracting and Normalizing Temporal Information. In Proc. Of IJCNLP'05, 694-706. (2005)
13. TIMEX2. In http://timex2.mitre.org/
14. TimeML. In http://www.timeml.org/site/index.html
15. Hersovici, M., Lempel, R., Yogev, S., Efficient Indexing of Versioned Document Sequences. In Proc. of ECIR. (2007)
16. Grandi, F., Introducing an Annotated Bibliography on Temporal and Evolution Aspects in the World Wide Web. SIGMOD Record 33(2), 84-86. (2004)
17. Anick, P., Flynn, R., Versioning a Full-Text Information Retrieval System. In Proc. Of SIGIR. (1992)
18. Zobel, J., Moffat, A., Inverted Files for Text Search Engines. ACM Comput. Surv., Vol.38(2), 6. (2006)
19. Nørvag, K., Nybø, A., DyST: Dynamic and Scalable Temporal Text Indexing. In Proc. Of TIME. (2006)
20. Stack, M., Full Text Search of Web Archive Collections. In Proc. Of IWAW. (2006)
21. Kumar, V., Tsotras, J., Faloutsos, C., Designing Access Methods for Bitemporal Databases. IEEE Trans. Knowl. Data Eng, Vol.10(1), 1-20 (1998)
22. Beckmann, N., Kriegel, H., Schneider, R., Seeger, B., The R-tree: An efficient and robust access method for points and rectangles. In Proc. Of SIGMOD, 322-331. (1990)
23. Nørvag, K., Space-Efficient Support for Temporal Text Indexing in a Document Archive Context. In Proc. Of ECDL, 511-522. (2003)
24. Nørvag, K., Supporting temporal text-containment queries in temporal document databases. Data Knowl. Eng., Vol.49(1), 105-125. (2004)
25. Nørvag, K., Nybø, A., Improving Space-Efficiency in Temporal Text-Indexing. In Proc. Of DASFAA, 791-802. (2005)
26. Nørvag, K., Nybø, A., DyST: Dynamic and Scalable Temporal Text Indexing. In Proc. Of TIME, 204-211. (2006)
27. Berberich, K., Bedathur, S., Neumann, T., Weikum, G., FluxCapacitor: Efficient Time-Travel Text Search. In Proc. Of VLDB, 1414-1417. (2007)
28. Berberich, K., Bedathur, S., Neumann, T., Weikum, G., A Time Machine for Text Search. In Proc. Of SIGIR, 519-526. (2007)
29. Salzberg, B., Tsotras, V., Comparison of Access Methods for Time-Evolving Data. ACM Comput. Surv., Vol.31(2), 158–221. (1999)
30. Li, W., Wong, K., Yuan, C., A Model for Processing Temporal References in Chinese, In Proc. of Workshop on Temporal and Spatial Information Processing at ACL'01, 33-40 (2001)
31. Alonso, O., Gertz, M., Yates, R., Clustering and exploring search results using timeline constructions. In Proc. Of CIKM, 97-106. (2009)
32. Jin, P., Chen, H., Zhao, X., Yue, L., Hybrid Index Structures for Temporal-textual Web Search, In Proc. of APWeb, LNCS 6612, 271-277, (2011)

Peiquan Jin is an Associate Professor at the School of Computer Science and Technology, University of Science and Technology of China (USTC). He received his Ph.D. degree in computer science from University of Science and Technology of China (USTC) in 2003. After that he spent two years for his post-doc research in the Department of Electronic Engineering & Information Science, USTC. He was a Visiting Scientist at the University of Kaiserslautern, Germany, in 2009. His research interests include Web search, knowledge management, and databases. Dr. Jin is a member of ACM, and is an editor of International Journal of Information Processing and Management and International Journal of Knowledge Society Research. He serves as a PC member of many international conferences, including DEXA'09-11, WAIM’11, NCM'08-11, and NDBC'09-11.

Hong Chen is currently a third-year master student at the School of Computer Science and Technology, University of Science and Technology of China. Her main research interests include Web information extraction and retrieval.

Xujian Zhao is currently a PhD candidate at the School of Computer Science and Technology, University of Science and Technology of China. His research interests are focused on temporal information retrieval in the Web, especially on Web-based news search. He spent six months in Alibaba in 2010 for e-business data mining work, and was a visiting scientist in University of Saarland, Germany, in 2011. He has published about ten papers in peer-reviewed conferences and journals.

Xiaowen Li is currently a third-year master student at the School of Computer Science and Technology, University of Science and Technology of China. Her main research interests include temporal information extraction and time-aware ranking for Web pages.

Lihua Yue is a full professor at the School of Computer Science and Technology, University of Science and Technology of China. She received his master degree in computer science in 1994 from University of Science and Technology of China. Her research interests are mainly concentrated on database and information systems. She is a member of ACM and a senior member of China Computer Federation (CCF). She has published more than 50 papers in peer-reviewed conferences and journals.

Received: April 07, 2010; Accepted: March 14, 2011.