Archiving Temporal Web Information: Organization of Web Contents for Fast Access and Compact Storage

(Technical Report UMIACS-TR-2008-08)

Sangchul Song and Joseph JaJa
Department of Electrical and Computer Engineering
Institute for Advanced Computer Studies
University of Maryland, College Park

Abstract

We address the problem of archiving dynamic web contents over significant time spans. Current schemes crawl the web contents at regular time intervals and archive the contents after each crawl regardless of whether or not the contents have changed between consecutive crawls. Our goal is to store newly crawled web contents only when they are different than the previous crawl, while ensuring accurate and quick retrieval of archived contents based on arbitrary temporal queries over the archived time period. In this paper, we develop a scheme that stores unique temporal web contents in containers following the widely used ARC/WARC format, and that provides quick access to the archived contents for arbitrary temporal queries. A novel component of our scheme is the use of a new indexing structure based on the concept of persistent or multi-version data structures. Our scheme can be shown to be asymptotically optimal both in storage utilization and insert/retrieval time. We illustrate the performance of our method on two very different data sets from the Stanford WebBase project, the first reflecting very dynamic web contents and the second relatively static web contents. The experimental results clearly illustrate the substantial storage savings achieved by eliminating duplicate contents detected between consecutive crawls, as well as the speed at which our method can find the archived contents specified through arbitrary temporal queries.

1. Introduction

The web has become the primary medium where new information concerning almost every facet of human activity is published. However the web is an ephemeral medium whose contents are constantly changing and new information is rapidly replacing old information, resulting in the disappearance of a large number of web pages every day. Therefore important web information should be archived on a regular basis to prevent the permanent loss of part of our cultural and scientific heritage. Many current efforts are trying to develop methodologies and tools for capturing and archiving some of the web's contents that are deemed critical. While many challenges for archiving web contents exist, we focus in our work on the problem of organizing and indexing archived web contents so as to minimize the amount of storage used and to enable fast retrieval of archived contents subject to arbitrary temporal web queries.

The Internet Archive [3], the world's largest web archive, has been the leader in developing methodologies and standards for archiving web contents. Their main goal is to capture significant snapshots over time of the whole web and to archive the corresponding contents. At this point, they hold around two petabytes of data and are growing at the rate of 20 terabytes per month. Their overall strategy consists of regular crawls of the web, followed by downloading the crawled contents into containers after each crawl and indexing the archived contents to allow future access based on URLs and crawl times.

We consider in this paper the problem of storing and indexing web contents using the crawling strategy but avoiding the storage of any duplicate web contents examined between two consecutive crawls. We develop a new scheme that achieves precisely this goal while ensuring quick access to the archived contents based on a temporal query covering any period during the archival time span. The scheme involves a novel indexing structure based on the concept of multi-version B-tree and a very efficient duplicate detection algorithm.

In the next section, we give an overview of some of the current strategies used to archive web contents, focusing on the format and scheme used by the Internet Archive, while Section 3 is devoted to a description of our scheme. Section 4 describes the experimental tests conducted on two significant web archives using our scheme and reports on its performance compared to the previous best known scheme used by the Internet Archive.

2. Archiving Containers

In this section, we examine a number of methods used to store web objects into an archive, focusing on the emerging standard for using containers as the storage unit for web archives.

2.1 Storage Types

To organize and store web objects in an archive, several methods have been proposed and are currently in use. A straightforward method (such as the one implemented in [1]) and is based on using a local file system where the target web material is copied object by object into the local file system, maintaining the relative structure among the objects. For future access, the html tag 'file' can replace the 'http' tag in the original object. We can then use the local file system for navigation through the archived web material. For example,'http://www.example.org/index.html' can be rewritten as 'file:///archive/2007.08.01/www.example.org/index.html'. These locally stored objects can be re-published through a web server for public web access (The National Library of Australia's Pandora project takes this approach [10]). It is relatively easy to set up and run this type of web archiving storage and the retrieval process is carried out using local file access mechanisms. However, there are several problems with this method including its limited scalability to what the local file system can handle, and more importantly it runs counter to a major guiding principle for long term archiving, namely platform independence. This strategy captures no metadata beyond the URL and date, and requires modifications to the original contents, and thus the strict faithfulness to the original contents cannot be maintained in most cases [12].

The second approach extracts documents from the hypertext context and reorganizes them in a different format while setting up different access mechanisms. For example, a small set of web pages can be converted into a single PDF document. However, this strategy makes sense only for specific objects that were originally created independently of the web. Although it is possible to maintain the hypertext structure *within* the converted documents, for the broader range archiving, this approach loses the hypertext structure between multiple such documents.

The most popular method currently in use by many web archives, including the Internet Archive [3], stores web objects in containers in a well-defined structure. A container holds a set of harvested web files, each with its own auxiliary metadata. The size of a container can vary up to hundreds of megabytes (usually 100~500MB). Typically, an external index is maintained to provide the mapping between hyperlinks inside a container and the locations of the archived objects that the hyperlinks point to. For example, if, inside a container, there is a web page archived on September 24, 2007 which has an outgoing hyperlink with a tag <a href="http://www.example.org/images/welcome.jpg>, the index could return in response to the tag and date the container ID and the offset within the container where the corresponding web page is stored. One of the most widely used container format is the ARC file format [6] that was originally developed by the Internet Archive and adopted by many others. Recently, building on the ARC format, an international collaborative effort developed a standard format called the WARC file format [4]. Upon the approval of the new standard format, many who have been either using their own container formats (such as the Stanford WebBase project [11]) or not taking the container approach (such as the Pandora project [10]) have agreed to convert to the standard format in the future. In our work, we will also exclusively use the ARC/WARC format.

In the next section, we will examine how the Internet Archive uses this format to index and access archived web data.

2.2 The Internet Archive and the Wayback Machine

The Internet Archive (IA) is a non-profit organization whose mission is to build a public Internet digital library. As of this writing, the Internet Archive claims to hold 2 petabytes of data and is growing at a rate of 20 terabytes per month [2]. The Wayback Machine is the access point to its archive of snapshots of the web, allowing users to access archived versions of web pages across time. With the Wayback Machine, users enter an URL and the Wayback Machine displays a list of all instances of the archived URL, marked by the crawl dates. Selecting a date begins browsing a site as it appeared at that crawl date, and continued navigation pulls up the linked pages archived on the same or closest available date.

The Internet Archive stores archived web pages in containers in the ARC format introduced in 1996 [6]. The standard WARC format is based on the ARC format with substantial overlap. In essence, an ARC file is a self contained file that aggregates multiple web objects returned through a variety of protocols, including http, ftp, news, gopher, and mail. The contained web objects are sequentially stored, each annotated by its URL record that contains the URL, IP-address, archive-date, content-type, result-code, checksum, length, and so on.

Although an individual ARC file is self-contained, almost all web archives set up and manage external indices that map a URL and crawl time to the ID of the container and the offset where the corresponding information is archived. Without an external index, a sequential look up through all ARC files to search for the web information will take an enormous amount of time for any significant size web archive.

Upon access, a portion of an ARC file that contains the requested content is delivered to users, however, with a JavaScript snippet attached in the end of the content on the fly. The purpose of the snippet is to rewrite hyperlinks within the content, so that they link to an archived object in the archive, not a current web page. For example, an outgoing link to http://www.yahoo.com in an archived object crawled in Feb. 29 2000 is altered as http://www.archive.org/web/20000229123340/http://www.yahoo.com. When the Wayback Machine receives an HTTP request to the rewritten URL, it looks up the external index that maps the URL to the location information (container ID and offset) in the archive, and returns the requested content (also with the JavaScript snippet embedded in it).

The next subsection further discusses how such external indices are maintained.

2.3 Container Indexing Schemes

The current indexing schemes used to map URLs to container IDs and offsets belong to two classes: sequential or structured. The first type consists of a sequential listing of URLs, each with the corresponding container information. The sequential index usually relies on external sorting and other data arrangement algorithms before records are written on a final index file. This implies a batch-mode indexing, where during each batch we have to acquire all the data to be indexed. Once sorted, searching for an entry requires $O(\log_2 N)$ where N is the number of entries in the batch file. Note that external sorting is in general an expensive operation, especially when we have to deal with a large index file. On the other hand, structured schemes are generally more sophisticated data structures, typically a B-Tree or one of its variants such as B^+-Tree. A B-Tree is a tree data structure where internal nodes store {key, data} records in increasing order according to the keys in the records. Each internal node also contains pointers to its children, and the keys separate the ranges of keys in each subtree. That is, all the keys in the first subtree are smaller than the first key of the parent node, and all the keys in the second subtree are larger than the first key and smaller than the second key of the parent node, and so on. In a B^+-Tree, in contrast to a B-tree, all records are stored at the lowest level of the tree; only keys are stored in interior nodes. For more details on these data structures, see [7]. The B-Tree and B^+-Tree support fast dynamic insertion and deletion of index entries as well as fast querying. In practice, databases are often involved as middleware to accommodate structured schemes. As of this writing , an open source implementation of the Wayback Machine [5] supports both the file-based sequential and B^+-Tree-based structured indices.

To accommodate both the URL and time indices, the Wayback Machine concatenates the URL and the crawl time, and uses the result as a key. Although this scheme is very easy to implement, and works well to handle queries with a specific URL and time, it has significant limitations when it comes to handling time slice or time span queries. For example, given a specific date or time span, a query to retrieve all the web pages of interest cannot be handled efficiently using this scheme. Moreover, this structure will have a separate entry every time a web page is crawled, even if the page has not changed since the last crawl.

A possible alternative is to set up two indexing structures, one using the URLs as keys and the second using the crawl times as keys. Two separate searches are performed on the indexing structures, and the results are then matched to find the final result. As the archive grows, the time it takes to combine the results of the separate searchers grows rapidly, and response time will suffer substantially.

We next describe our scheme that stores a single copy for all the duplicate web objects crawled at consecutive times, and that can handle exact time queries as well as time slice and time span queries quite fast in theoretically optimal time.

3. Handling Temporal Web Information

An important characteristic of a web archive is the fact that it typically represents a collection of temporal snapshots of web objects. As an extreme case, the archiving of a commercial news website over some time span will have to capture all the different versions of the website as the news are updated many times on a daily basis. However most of the information on such a website will change less frequently, and hence archiving the whole site separately every time a small piece of news has been updated is extremely wasteful. In general, it is not easy to detect the exact time when a target website is updated, and the strategy commonly used is to let web crawlers visit the target website repeatedly, archiving the website separately each time it is crawled. In our scheme, we only store the web pages that have changed since the last crawl while being able to retrieve and reconstruct the web information present at any time specified by the user.

We next introduce our scheme, starting with our novel version-aware persistent data structure, called PISA (Persistent Indexing Structure for Archives). Here we give only an overview of this structure and summarize its performance. Detailed description of PISA and the algorithms for handling various operations are given in [13].

3.1 The PISA Indexing Structure

Previous indexing schemes construct a B^+-Tree in which each key entry consists of the concatenation of a URL and crawl time. Based on the concatenated key, a search through the B^+-Tree leads to a container ID and an offset where the corresponding content has been stored. Clearly the B^+-Tree will have separate entries for the same URL with different crawl times, and the corresponding web content will be stored separately for every crawl even though the content may not change over the past several crawls. To avoid storing any duplicate web content and yet maintain the ability to recover the contents corresponding to a URL and a date, we use techniques that are similar to those introduced for persistent or multi-version data structures [8, 9, 14]. Our PISA indexing structure is somewhat simpler and more practical than those reported in the literature, but achieves the same asymptotic performance for our dynamic operations as the best known schemes.

Rather than dealing with a key consisting of a URL and crawl time, our key consists of a URL and a time interval [birth-time, end-time] during which the corresponding web content has not changed. The birth-time corresponds to the time when a particular content under this URL was first seen, and the end-time corresponds to the earliest crawl time when the corresponding content changes. In addition to the key and the time interval, we include the hash value of the corresponding content. This value is used to check whether the content has changed since the last crawl. We build a B^+-Tree in which the leaves contain the mapping information of each key to the corresponding container and offset. PISA maintains the B^+-Tree structure while inserting the new information after a crawl to a URL. If the URL is already there with a "live" time interval (that is, time interval is open with no end time), we check whether the content is the same as the last time it was crawled (using the saved hash value) in which case we are done. Otherwise we have to insert a new entry into PISA with an open time interval starting with the current crawl time. In addition, a copy of the content has to be saved in a container. To achieve optimal performance, each of the query and update operations on PISA has to be performed in time logarithmic in the total number of entries, while retaining simplicity for practical implementation. PISA does indeed achieve this as detailed in [13].

We illustrate the benefits of PISA through the following example. Figure 1 depicts an example of archiving a web page (at URL A) over a period of time. In the figure, there have been two updates at t_3 and t_9, and nine visits have been made from an archiving crawler at t_1, t_2, t_4, t_5, t_6, t_8, t_{10}, t_{11} and t_{12}. In a conventional web archive, the content crawled at each visit, whether or not duplicated, is archived in its storage to keep track of the web page's history as detailed as possible. If the archive discarded any duplicates (those at t_2, $t_{5\sim8}$ and $t_{11\sim12}$), and stored only unique contents (those at t_1, t_4 and t_{10}), it would not be able to deliver the best content to the user. For example, consider an example where a user wants to view the web page as it appeared at t_7. With the full archiving with duplicates, the archive would return the correct content – Content 2. However, if the archive had only stored the content at t_1, t_4, and t_{10}, the archive would have no idea that it actually saw Content 2 at a later time (t_8), and it would have to make a wild guess between Content 2 and Content 3, probably ending up returning the closer (but wrong) one – Content 3.

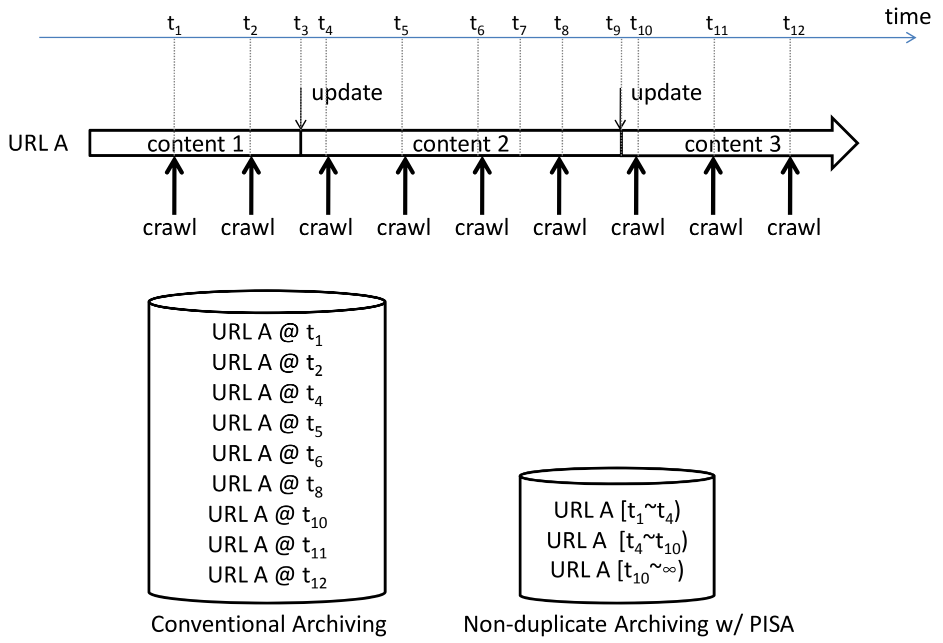

FIGURE 1. COMPARISON BETWEEN CONVENTIONAL ARCHIVING AND ARCHIVING WITH PISA

On the other hand, our methodology eliminates the need of storing duplicates without compromising the correctness of the retrieved contents. Rather than indexing records using their URLs concatenated by the timestamp as a key, PISA maintains the lifespan during which each record was alive, and provides operations to query and insert into the data structure very efficiently. In fact, the speeds of those operations are effectively much faster in PISA than in a conventional index since the total number of web objects that a non-duplicate archive has to maintain is clearly much smaller. For instance, in the previous example in Figure 1, for the same crawls, our methodology only needs to store three distinct contents. Since PISA maintains the information that Content 2 covers the time interval [t_4~t_{10}], a query for the content at t_7 can be confidently and correctly handled by returning Content 2, without having had to store all nine crawled contents.

One side benefit that PISA can also bring is that it can be also utilized in detecting duplicates. Each record in PISA can also contain a hash of the content that the record refers to, and since navigating through PISA for its record is much faster than through the archive store for the actual content, this hash can help determine the duplicates without much effort.

Another benefit of PISA is that PISA is capable of supporting extra operations that are considered prohibitive using conventional indices. For instance, pure-timeslice queries ("Given a time interval T, find all objects archived during this interval") cannot be practically supported in conventional indices, but can be performed efficiently in PISA. Moreover, approximate key-timeslice queries ("Given a key and a contiguous time interval T, find the objects with the closest key that were archived during interval T") can also be better supported in PISA.

4. Experimental Results

To illustrate the benefits of our scheme, we run two sets of tests. The first experiment is set up to show the impact of the duplicate elimination performed by PISA as well as to provide an estimate on the extra time required for duplicate elimination. The second experiment is performed to compare PISA to B^+-Tree, which represents the best previously used indexing structure for web archiving. To conduct our tests, we make use of the web archive from the Stanford WebBase project [11]. The crawled web pages were stored as the pages existed at the crawling time with no attempt to remove duplicates. Among the vast amount of data maintained by the archive, we choose the two datasets illustrated in Table 1 – one to represent fast-changing websites, the other to represent relatively static websites. The full lists of all the websites covered by each dataset are available in Appendix A.

TABLE 1. DATASETS

	# Crawls	Crawling Dates	Average # Pages / Crawl	Total # Pages	Average Size / Crawl	Total Size
News Websites	8	10/25/2006 ~ 11/2/2006	298,269	2,335,600	9.32GB	73.38GB
Governors Websites	28	09/08/2005 ~ 10/07/2005	10,055	273,717	228MB	5.98GB

The News Websites dataset is from a Stanford's event-based collection pertaining to the 2006 US House of Representatives Election, which was originally crawled from October 25, 2006 to November 11, 2006. In our experiments, we only use the data crawled on the first eight days. Since news websites tend to be updated very frequently (several times a day), an archive collection obtained by a series of daily crawls is less likely to contain duplicate web pages. Therefore, we expect that this dataset to be representative of an archiving scenario over fast changing websites, where duplicates are unlikely.

The second dataset is from a Stanford's event-based collection pertaining to Hurricane Katrina, originally crawled from September 3, 2005 to October 29, 2005. Since the total data size is too big to be accommodated on our machine, we extract the following part of the original collection. Within the collection, we consider 28-day crawls (September 8, 2005 to October 7, 2005, some intermediate dates are missing in the archive) over eight State Governors' websites (Alabama, Mississippi, North Dakota, New Mexico, Oklahoma, Pennsylvania, Texas and Louisiana), and perform the experiments on this dataset. Since updates in government websites are usually slow-paced, this dataset is representative of an archiving scenario over slow changing websites, where duplicates are expected to occur often.

Using these two datasets, we run our scheme and counted the number and size of duplicate web pages. We also monitor the time overhead incurred due to duplicate checking. The results from each data set are as follows.

For the News Websites dataset, we observe that, on average, 23% of crawled web pages – accounting roughly 12% in size – had not been changed. Considering the fast-changing nature of news websites, the amount of duplicates exceeds our initial expectation. This can be explained by the fact that, while the front pages were updated frequently, some of the content referred to on the front page remained the same. On the other hand, regarding the Governors' Websites, we find that, on average, 82% of crawled web pages – amounting approximately to 73% in size – were duplicates, which constitute a considerable portion of the crawled data. It is clear that our scheme will save a substantial amount of storage regardless of the type of the archived web data.

However, our scheme incurs an overhead due to the extra processing required for duplicate detection. We now provide estimates of this overhead based on our tests on the two data sets. In measuring the time overhead, we exclude the disk and network I/O time for fetching web objects, tasks that have to be done by any scheme, and focus on the overhead required for detecting duplicates.

For the News Websites dataset, we observe that the total time overhead to process 2,335,600 pages (73.38GB) was less than 8 minutes. This corresponds to 0.36 ms per page or 11.08 ms per MB. Therefore, the average processing speed for duplicate detection amounts to a processing rate of 90MB/s, which exceeds the typical disk I/O capacity. Similarly, the Governors' Websites dataset showed a time overhead of 0.1 ms per page or 4.44 ms per MB. The average processing speed per second amounts to 225MB/s, which is even faster than the News Website dataset. The results are summarized in Table 2. The details of the results can be found in Appendix B. Therefore, it can be seen that the additional steps required by our scheme only introduce a negligible time overhead.

TABLE 2. TIME OVERHEAD FOR DUPLICATE DETECTION

	Total	Per Page (Average)	Per MB (Average)
News Websites	13 m 52 s	0.356 ms	11.076 ms
Governors' Websites	0 m27 s	0.098 ms	4.444 ms

4.1 Performance Comparison: PISA vs. B$^+$-Tree

In this experiment, we compare the performance of PISA and B$^+$-Tree for inserting new information and for retrieving temporal web information from a web archive. Using the same Governors Websites archive, we measure the time that each of PISA and B$^+$-Tree spends in handling the same set of queries. In B$^+$-Tree, each key is created by concatenating the hash of a URL with the crawl time, whereas in PISA, the hash of a URL serves as a key, and a new entry is inserted with the key and the crawl time. Our comparison is based on running tests based on the operations listed in Table 3, which we believe capture the most important types of querying archived web contents.

TABLE 3. ACCESS OPERATIONS

Operation	**Description**
(URL, time) Insert	Given an object defined by a URL and a crawl time, insert the information regarding the object into the indexing structure with a pointer to its container and the offset within the container.
(URL, time) Query	Given a URL and a time, determine the location of the content of the URL that was valid at the specified time. The location refers to the container ID and the offset within the container.
(URL, time span) Query	Given a URL and a time span, determine the locations of the contents of the URL that were valid during the specified time span.
(URL, ---) Query	Given a URL, determine all the locations and crawl times corresponding to the URL.
(----, time span) Query	Given a time span, determine all the URLs and their corresponding locations in the archive, which were valid at the specified time span.

In B$^+$-Tree, a web page was considered valid at time t, if the web page was the latest page crawled earlier than t.

Our asymptotical analysis [13] shows that PISA can perform Insert, (URL, time), (URL, time span), and (URL, ---) queries in optimal time similar to B^+-Tree when the same number of entries are maintained in both structures. As for the (----, time span) query, PISA is asymptotically much faster than the B^+-tree.

We now examine the practical performance of the two structures in the context of the Governors Websites. This archive is indexed using both PISA and B^+-Tree. When inserting the input data, we measure the time taken for each indexing scheme to insert all the pages to be indexed. As we have seen before, this data set has 82% duplicate web pages, and as a result PISA took significantly less time than B^+-Tree (3.7s vs. 16.8s). This is an accumulated time which is not linear to the total number of inserted pages since an individual insert time becomes longer as the index gets larger.

As for the (URL, ----) Query, we perform the query for all the 27,643 distinct URLs present in the data set. In this experiment, the two indexing schemes finished the queries within almost the same time (2.7s vs. 3s).

The (URL, time span) Query is a special case of the (URL, ----) Query since it does not retrieve the entire history of a URL, but only pages that were valid during the specified time span. We perform (URL, time span) Query for each of 27,643 URLs for a time span of seven versions in the middle. More specifically, for each URL, we make a query to retrieve pages that were valid during September 19, 2005 ~ September 25, 2005. As in (URL, ----) Query, the two schemes showed similar results, finishing all the queries within 3 seconds.

On the other hand, the (----, time span) Query requires drastically different times for the two schemes. We perform 28 queries for each of 28 different crawl dates in the dataset. In this case, PISA handles all the queries within 1.6s, while the B^+-Tree takes 10.3 seconds. This can be explained by the fact that each of the entries in the B^+-Tree has to be examined before we can determine which objects to return.

So far, it is clear that PISA performs most operations at least as fast as B^+-Tree, while an operation such as (----, time span) Query can be performed much faster. However, PISA does incur an overhead in terms of space usage by the index (while saving substantially on the storage of the archived web contents). In fact, given the same number of entries in the index, PISA requires more disk blocks. In our experiment for the Governors Websites, PISA maintains 8,647 blocks whereas B^+-Tree has only 4,002 blocks. Assuming that each block takes about 2KB of storage space, PISA and B^+-Tree require 16.9MB and 7.8MB, respectively. Nevertheless, considering that the adoption of PISA has saved 4.3GB of storage space that, otherwise, would have been taken up by the duplicate web pages, the index space overhead is minimal. In fact, as the number of crawl dates increases, we expect the relative overhead of the PISA indexing structure to become smaller.

Table 4 provides a summary of our experimental results comparing PISA to B^+-tree.

TABLE 4. PERFORMANCE COMPARISON BETWEEN PISA AND B^+-TREE

	Time (ms)					Indexing Size (MB)
	Insert all pages	(URL, time) Query	(URL, ----) Query	(URL, time span) Query	(----, time span) Query	
PISA	3,717	4,814	2,824	2,677	1,642	16.89
B^+-Tree	16,767	7,313	2,966	2,968	10,285	7.82

5. Conclusion

We addressed in this paper the problem of archiving temporal web contents in such a way as to minimize the amount of storage required while being able to retrieve accurately any of the archived web information using arbitrary temporal web queries. Our new method uses the crawling strategy at regular time intervals, commonly used by all the known web archiving methods, but detects and eliminates the storage of duplicates between consecutive crawls. The archived web information is indexed using a novel indexing structure, called PISA – Persistent Indexing Structure for Archives, which enables quick duplicate detection and quick information discovery. We have presented in this paper experimental results on two significant web archives which illustrated the type of storage savings achieved by our scheme as well as its overall performance in retrieving archived data. It was also shown that the overhead incurred by our indexing structure is very minimal.

6. References

[1] HTTrack `http://www.webcitation.org/5SCSBqOXe`

[2] *The Internet Archive - Frequently Asked Question* `http://www.webcitation.org/5UvJ31WFD`

[3] *The Internet Archive - The Wayback Machine* `http://www.webcitation.org/5SCSL2r8e`

[4] *WARC, Web ARChive file format* `http://www.webcitation.org/5RPhvw0Wa`

[5] *Wayback* `http://archive-access.sourceforge.net/projects/wayback/`

[6] *WWW Archive File Format Specification* `http://pages.alexa.com/company/arcformat.html`

[7] R. Bayer and E. M. McCreight, *Organization and Maintenance of Large Ordered Indexes*, Acta informatica, 1 (1972), pp. 173-189.

[8] B. Becker, S. Gschwind, T. Ohler, B. Seeger and P. Widmayer, *An Asymptotically Optimal Multiversion B-tree*, The VLDB Journal, 5 (1996), pp. 264-275.

[9] B. Becker, S. Gschwind, T. Ohler, B. Seeger and P. Widmayer, *On Optimal Multiversion Access Structures*, *Proceedings of the Third International Symposium on Advances in Spatial Databases*, Springer-Verlag, 1993.

[10] W. Cathro, C. Webb and J. Whiting, *Archiving the Web: The PANDORA Archive at the National Library of Australia*, *the Preserving the Present for the Future Web Archiving Conference*, Copenhagen, Denmark, 2001.

[11] J. Hirai, S. Raghavan, A. Paepcke and H. Garcia-Molina, *WebBase : A repository of Web pages*, *The 9th International World Wide Web Conference (WWW9)*, Amsterdam, 2000.

[12] J. Masanès, *Web Archiving: Issues and Methods*, *Web Archiving*, Springer, Berlin, 2006, pp. 1-53.

[13] S. Song and J. JaJa, *A New Technique for Archiving Temporal Web Information*, *UMIACS Technical Report- In Preparation*, University of Maryland Institute for Advanced Computer Studies, 2008.

[14] P. J. Varman and R. M. Verma, *An Efficient Multiversion Access Structure*, IEEE Transactions on Knowledge and Data Engineering, 9 (1997), pp. 391-409.

Appendix A. List of Websites

News Websites

www.usatoday.com
online.wsj.com
www.nytimes.com
www.latimes.com
www.washingtonpost.com
www.chicagotribune.com
www.nydailynews.com
www.philly.com 10002
www.rockymountainnews.com
www.chron.com
www.nypost.com
www.freep.com
www.dallasnews.com
www.startribune.com
www.boston.com
www.nj.com
www.ajc.com
www.azcentral.com
www.newsday.com
www.sfgate.com
www.cleveland.com
seattletimes.nwsource.com
www.stltoday.com
politics.tampabay.com
www.signonsandiego.com
www.jsonline.com
www.baltimoresun.com
www.miami.com
www.oregonlive.com
www.post-gazette.com
www.kansascity.com
www.ocregister.com
www.dispatch.com
www.indystar.com
www.mysanantonio.com
www.OrlandoSentinel.com
www.sun-sentinel.com
www.sacbee.com
www.dfw.com

Governors Websites

www.governor.alabama.gov
www.governorbarbour.com
www.governor.state.nd.us
www.governor.state.nm.us
www.governor.state.ok.us
www.governor.state.pa.us
www.governor.state.tx.us
www.gov.state.la.us

Appendix B. Experiment Results

TABLE 5. DUPLICATES IN NEWS WEBSITES

	# Web	Total	# Duplicates	Duplicate	% Duplicates	% Duplicates
10/25/2006	247718	8.11	0	0	0	0
10/26/2006	258283	8.35	58303	0.97	22.57	11.67
10/27/2006	273658	8.81	63466	1.04	23.19	11.77
10/28/2006	309177	9.54	71481	1.32	23.12	13.88
10/30/2006	310234	9.41	74543	1.17	24.03	12.41
10/31/2006	299193	9.16	70296	1.04	23.50	11.36
11/1/2006	322836	10.05	75442	1.26	23.37	12.53
11/2/2006	314501	9.95	64135	0.95	20.39	9.60
Total	2335600	73.38	477666	7.76	20.45	10.57
Average	298268.86	9.32	68238	1.11	22.88	11.89

TABLE 6. DUPLICATES IN GOVERNORS' WEBSITES

	# Web	Total Size	# Duplicates	Duplicate	% Duplicates	% Duplicates
9/8/2005	5245	108.77	0	0.00	0.00	0.00
9/9/2005	5261	109.38	4482	73.67	85.19	67.35
9/11/2005	5273	109.69	4600	75.36	87.24	68.71
9/12/2005	5275	110.04	4600	75.37	87.20	68.50
9/13/2005	5281	110.16	4596	75.10	87.03	68.17
9/14/2005	5298	111.11	4603	75.19	86.88	67.66
9/15/2005	5313	111.86	4080	69.22	76.79	61.89
9/16/2005	5322	111.85	4577	75.07	86.00	67.11
9/17/2005	5339	112.58	4611	75.50	86.36	67.07
9/18/2005	5065	109.28	4364	73.31	86.16	67.08
9/19/2005	5357	112.61	4644	76.15	86.69	67.62
9/20/2005	5356	112.81	4531	74.11	84.60	65.70
9/21/2005	5401	114.35	4565	74.62	84.52	65.25
9/22/2005	5385	113.93	4596	75.11	85.35	65.93
9/23/2005	2148	59.21	1398	19.19	65.08	32.41
9/24/2005	5449	116.53	4604	74.93	84.49	64.30
9/25/2005	5453	115.84	4664	76.37	85.53	65.93
9/26/2005	2194	59.30	1419	19.51	64.68	32.90
9/27/2005	5481	116.69	4575	74.77	83.47	64.08
9/28/2005	5492	117.35	4126	69.77	75.13	59.46
9/29/2005	22421	506.82	4621	75.28	20.61	14.85
9/30/2005	22427	506.04	20157	422.85	89.88	83.56
10/2/2005	22265	504.65	19824	417.89	89.04	82.81
10/3/2005	22270	504.52	21094	451.01	94.72	89.39
10/4/2005	22289	505.83	20188	427.65	90.57	84.54
10/5/2005	21727	483.93	19068	399.16	87.76	82.48
10/6/2005	22342	505.88	20784	463.19	93.03	91.56
10/7/2005	20588	465.52	18516	414.23	89.94	88.98
Total	276717	6126.51	223887	4373.56	80.91	71.39
Average	10054.52	222.88	8292.11	161.98	82.37	66.86