# Managing duplicates in a web archive

Daniel Gomes
Universidade de Lisboa
1749-016 Lisboa, Portugal
dcg@di.fc.ul.pt

André L. Santos
Universidade de Lisboa
1749-016 Lisboa, Portugal
als@di.fc.ul.pt

Mário J. Silva
Universidade de Lisboa
1749-016 Lisboa, Portugal
mjs@di.fc.ul.pt

## ABSTRACT

Crawlers harvest the web by iteratively downloading documents referenced by URLs. It is frequent to find different URLs that refer to the same document, leading crawlers to download duplicates. Hence, web archives built through incremental crawls waste space storing these documents. In this paper, we study the existence of duplicates within a web archive and discuss strategies to eliminate them at storage level during the crawl. We present a storage system architecture that addresses the requirements of web archives and detail its implementation and evaluation. The system is now supporting an archive for the Portuguese web replacing previous NFS-based storage servers. Experimental results showed that the elimination of duplicates can improve storage throughput. The web storage system outperformed NFS based storage by 68% in read operations and by 50% in write operations. [1]

## 1. INTRODUCTION

Archiving the web is crucial for preserving cultural heritage. However, given the large amount of data involved, web archiving is a costly and complex task that claims for specific software to do it automatically. The growing interest in web archives raised the need for storage systems able to address the specific characteristics of web document collections. It is difficult to avoid downloading duplicates during the crawl of a large set of documents because they are commonly referenced by distinct and apparently unrelated URLs [2, 25, 28]. Thus, duplication of documents is prevalent in web collections and must be eliminated at storage level. Duplicates are caused by the replication of documents to ensure availability of the information, use of other sites as templates or web servers that present error pages with identical content when a given resource accessed through them is unavailable.

A collection of web documents built incrementally presents an additional number of duplicates due to the documents that remain unchanged. An approach to solve this problem is to estimate the frequency of change of pages based on historical data, reducing the probability of an incremental web crawler downloading a page that has not changed since the last crawl [11]. Unfortunately, historical data for most web resources is not available and the lifetime of URLs that reference pages is short because new URLs are permanently being created while others disappear [17, 39]. The elimination of duplicates at storage level after a crawl is terminated can be executed in batch by finding and deleting duplicates. However, disk IO operations are expensive and this approach imposes that every document is written on disk during the crawl, then read to be compared with others to identify duplicates and deleted if it is a duplicate. Considering that a web archive holds millions of documents and new ones are frequently being loaded, this approach can be prohibitively time consuming. The elimination of duplicates is an appealing feature to save storage space but it must not jeopardize the system's performance.

In this paper we study the presence of duplicates within a web archive and propose a mechanism to detect if a document is a duplicate before it is stored on disk. We present an algorithm for the elimination of exact duplicates at storage level, which does not overload the crawler with additional data structures, and discuss its adequation for web archiving. We validated the algorithm by applying it to the implementation of Webstore, a system designed to enable the storage and maintenance of web documents. We describe the use of Webstore as the storage manager of a Portuguese web archive.

This paper is organized as follows: Section 2 presents a preliminary study of document duplication and URL persistence within a web archive. In Section 3 we discuss the elimination of exact and partial duplicates within a web archive. Section 4 proposes an algorithm to eliminate exact duplicates based on fingerprint signatures and discusses techniques to reduce the probability of loosing documents due to the occurrence of fingerprint collisions. Section 5 presents the motivation for the development of Webstore and describes its requirements and implementation. In Section 6, we present and discuss experimental results. In Section 7, we present related work. Finally, in Section 8, we draw our conclusions and propose directions for future work.

[1]This study was partially supported by the FCCN-Fundação para a Computação Científica Nacional, FCT-Fundação para a Ciência e Tecnologia, under grants POSI/SRI/40193/2001 (XMLBase project) and SFRH/BD/11062/2002 (scholarship).

| Id | Date | # URLs | % dups. in crawl | % dups. last crawl | % URLs persistent |
|---|---|---|---|---|---|
| 1 | 07-2002 | 1.6M | 23 | - | - |
| 2 | 10-2002 | 1.2M | 21.4 | 7.4 | 36 |
| 3 | 03-2003 | 3.5M | 15.4 | 9.6 | 14 |
| 4 | 10-2003 | 3.3M | 10.9 | 18.5 | 27 |
| 5 | 06-2004 | 4.4M | 6.7 | 18.1 | 32 |

**Table 1: Duplication found in 5 crawls performed by the tumba! search engine.**

## 2. DUPLICATION AND URL PERSISTENCE WITHIN A WEB ARCHIVE

We measured duplication and URL persistence through the analysis of 5 crawls stored in an archive of the Portuguese web that includes documents written in Portuguese hosted on sites under several domains [20]. Each crawl was seeded with the home pages of the sites harvested in the previous crawl. We computed the MD5 digests obtained for every crawled document to detect duplicates. In the presence of duplicated documents gathered from the web, it is not possible to automatically identify which URL is a replica and which is the original one. So, when several URLs point to the same document, we elect randomly one of the URLs as the reference to the original document and the remaining as duplicates. Table 1 summarizes the obtained results. For each crawl, we present the date in which the crawl was started, the total number of URLs visited, the percentage of documents that were duplicates within the crawl and the percentage of documents and URLs that already existed in the previous crawl. We identified that over 25% of the documents kept in the archive were exact duplicates. The first version of the crawler generated crawl *1*, where *23%* of the documents were duplicates. The number of duplicates within crawls has been decreasing due to consecutive optimizations of the crawler, such as spider trap detection mechanisms. We did not find similar studies on web archives. However, Shivakumar and Garcia-Molina counted 18% of pages having an additional replica in their experiments using data crawled for an earlier version of the Google search engine [36] and Mogul identified that 16.3% of documents were duplicates when analyzing a proxy trace [28]. Heydon and Najork identified that 8.5% of the documents fetched were duplicates when using the Mercator crawler [23]. We think that the differences between the levels of duplication are due to different usage contexts and experimentation. The results obtained show that a web archive presents higher duplication levels than other applications due to the incremental construction of the document collection. On the $5^{th}$ column of Table 1 we can observe that a significant number of documents remains unchanged between crawls. Although pages often change, documents such as research articles are static.

On the rightmost column the number of URLs that existed on the previous crawl is relatively low, suggesting that new URLs are continuously being created and older ones quickly disappear. The problem of URL persistence to reference archived documents could be resolved through the usage of Universal Resource Names (URNs) registered and maintained by the holders of the Fully Qualified Domain Name registration [15]. But so far, the existence of URNs is insignificant. National Libraries tried to maintain URNs to selected publications but the human intervention required, made URNs unbearable at web scale [29]. A study on the persistence of URLs cited in research articles showed that after four years 40–50% of the referenced URLs become inaccessible [39]. Our results show that the decay of URLs is much faster on broader collections than research articles. Fetterly et al. witnessed in several weekly crawls of a set of 150 million URLs that after 11 weeks, 12% of the URLs became unavailable [17]. The URLs analyzed were gathered by crawling documents linked from the Yahoo! home page using a web traversal strategy biased towards pages with high PageRank. We believe that the number of URLs that disappeared was relatively small because the URLs of important pages tend to be more persistent. On our turn, the pages analyzed in the web archive were gathered from exhaustive crawls of the Portuguese web regardless their importance.

## 3. PARTIAL VS. EXACT DUPLICATES

When a set of documents are bytewise equal we say that they are exact duplicates. Documents that replicate a pert of the content are partial duplicates. Delta storage or encoding, is a technique used to save space that consists on storing only the difference from a previous version of a document [27]. There are web pages that suffer only minor changes overtime, such as the number of visitors received or the current date. Delta storage enables to store only the part of the document that has changed, eliminating partial duplicates. Versioning systems (e.g. CVS [1]) save disk space by using delta storage. They assume that objects change in time maintaining a descendence tree under an unique identifier (file name). However, duplication among pages referenced by different identifiers (URLs) is prevalent on the web and using delta storage to archive them causes continuous duplication in independent trees. Plus, each tree holds few versions of a document because URLs are highly transient. The reorganizations in the directories of a site or changes of domain cause the disappearance of URLs without implying changes on its documents. For instance, recently a very large number of sites of the Portuguese public administration changed their domains from .PT to sub-domains of .GOV.PT without further changes on the documents provided.

Delta storage also raises preservation issues, because the retrieval of a document kept in delta format must be rebuilt by traversing the descendence tree. These algorithms are software dependent and a corruption on the tree may turn the subsequent versions of the document inaccessible. Moreover, delta storage algorithms can only process a limited set of document formats. A delta storage algorithm that processes an HTML file can not be applied to a Macromedia Flash movie, so it is highly dependent on technological changes of the formats used for web publishing. Delta storage imposes expensive comparisons to detect partial duplications between the last document in the descendence tree and the document to store. These comparisons executed while documents are being intensively loaded could originate a serious bottleneck in a web archive.

The elimination of partial duplicates to save storage space was analyzed in two studies by You and Karamanolis [40] and, Denehy and Hsu [16]. In the first study the authors compared intra-file compression against inter-file compression using delta encoding and data chunking [40]. These techniques are used to save storage space within a set of documents by eliminating partial duplicates. However, they require additional meta-data to reconstruct the original documents, which may become a preservation problem if the meta-data is lost. You and Karamanolis also evaluated several techniques with distinct data sets and concluded that none of them presented optimal results for all data sets. For the data set containing web pages, compressing each page independently causes just an increase of 4% over the best result presented (using delta encoding). Given the heterogeneity of the document formats found on the web, we conclude that a web storage system should support independent compression algorithms according to the document's format. The authors evaluated the compression factor of each one of the techniques without measuring performance but described performance problems in distributed system architectures. In the second study, Denehy and Hsu evaluated different methods for detecting partial duplicates and proposed a system that stores unique blocks of data with a configurable replication level [16]. The evaluations were mainly ran over an email corpus. Although emails contain attachments of formats commonly found on the web, it is arguable infer that the best method (sliding window) would be as efficient in a web corpus. The prototype presented a centralized architecture. Both studies suggested that eliminating partial duplicates in distributed systems raises new problems. However, web archives require distributed storage systems to face the large amounts of web data they must hold. The detection of partial duplication has been also studied to detected plagiarized documents yielding good accuracy results [3, 18, 35]. However, the presented methods are too heavy to be applied among large collections of documents.

Based on the results obtained in our experiments and related work, we conclude that duplication is considerable within a web archive but eliminating partial duplicates is not recommended because the existent storage systems assume the persistence of document identifiers, can not be efficiently applied in large-scale distributed storage and raise preservation issues. We believe that the elimination of exact duplicates through a lightweight mechanism based on document fingerprints is more suitable. Nevertheless, eliminating partial duplicates could be useful to save space while storing small crawls of selected URLs executed with short intervals of time.

## 4. MANAGING DUPLICATES

### 4.1 Elimination of exact duplicates

In this Section, we describe the data model and algorithms involved in the duplicates elimination mechanism. The data model relies on 3 main classes: *instance*, *volume* and *block*. The instance class provides a centralized view of a storage space composed of volumes containing blocks. Each block keeps a document and related operational meta-data. The *signature* is the number obtained from applying a fingerprinting algorithm to the document. A *contentkey* contains the signature of the document and the volume where it was stored. A *block* holds an unique document within the volume. It is composed by a *header* and a *data container* (Figure 1). The data container keeps the document. The header contains information about the software version, the document's original size in bytes and a reference counter that keeps track of the difference between the storage and delete

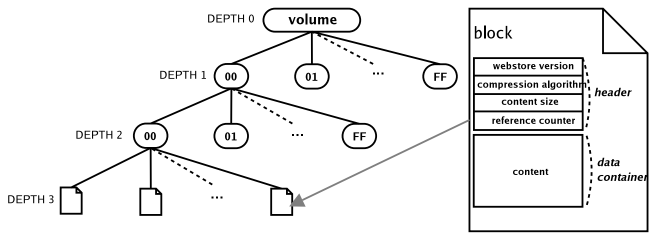

**Figure 1: Storage structure of a volume: a tree holding blocks on the leafs.**

requests performed on the document, allowing independent applications to share the same instance without interfering with each others data processing. The header also specifies the algorithm used to compress the document, allowing the coexistence of several compression types within the volume and the application of suitable algorithms according to the document's format. The storage structure of a volume is a tree containing blocks on its leafs. Figure 1 illustrates a storage structure with depth 3. The nodes within each level of depth are identified by numbers represented in hexadecimal format from 0 to FF. The tree depth can change within the volumes that compose an instance, according to the storage capacity of the node.

The location of a block within the volume tree is obtained by applying a function called *sig2location* to the document's signature. Assuming that the signature of a document is unique, two documents have the same location within a volume if they are duplicates. Consider a volume tree with depth $n$ and a signature with $m$ bytes of length. Sig2location uses the *(n - 1)* most significant bytes in the signature to identify the path to follow in the volume tree. The $i^{th}$ byte of the signature identifies the tree node with depth $i$. The remaining bytes of the signature (*m-n-1*) identify the block name on the leaf of the tree. For instance, considering a volume tree with depth *3*, the block holding a content with signature *ADEE2232AF3A4355* would be found in the tree by following the nodes AD, EE and leaf 2232AF3A4355.

The detection of duplicates is performed during the storage of each document, ensuring that each distinct document is stored in a single block within the instance. When a client requests the storage of a document, the system performs a sequence of tasks:

1. Generates a signature $s$ for the document;

2. Applies sig2location to the signature and obtains the location $l$ of the corresponding block;

3. Searches for a block in location $l$ within the $n$ volumes that compose the instance, multicasting requests to the volumes;

4. If a block is found on one of the volumes, the document is considered to be a duplicate and its reference counter is incremented. Otherwise, the document is stored in a new block with location $l$ in the volume identified by $s$ *mod* $n$;

5. Finally, a *contentkey* referencing the block is returned to the client.

The mod-based policy used to determine the volume where to store a document divides the load equally among the volumes. However, clients can define the volume where to store each document implementing alternative load-balancing policies. This way, in the presence of heterogeneous nodes, clients can impose higher workloads on the ones with higher throughput.

A client *retrieves* a document by supplying the corresponding contentkey. Webstore decomposes the contentkey, identifies the signature of the document and the volume that hosts the correspondent block. The location of the block in the volume is obtained by applying sig2location to the signature. Finally, the document stored in the block is decompressed using the algorithm specified in the block's header, and the document is returned to the client.

The *delete* operation is also invoked with a contentkey as argument. The location of the block is executed by following the same process as for the retrieve operation. If the reference counter contained in the header of the block has value 1, the block is deleted. Otherwise, the reference counter is decremented. Since the location of the document is determined by the contentkey, the volume where the document is stored is directly accessed, both for the retrieve and delete operations. Therefore, the performance of these operations is independent from the number of volumes that compose an instance.

### 4.2 Fake duplicates

Theoretically, if two documents have the same signature they are duplicates. However, fingerprinting algorithms present a small probability of *collision* that causes the generation of the same signature for two different documents [32]. Relying exclusively on the comparison of signatures to detect duplicates within a large collection of documents, could cause some documents to be wrongly identified as duplicates and not stored. We call these situations *fake duplicates*. The probability of occurrence of collisions depends on the fingerprinting algorithm but it is extremely small in most cases (less than one in one million). On the other hand, a web archive holds millions of documents so a fake duplicate may occur. The remote possibility of losing a document is acceptable for most people, but it is disquieting for a librarian in charge of preserving an unique document of great historical importance. We believe that the probability of loosing a document due to a disk error or bug on the underlying software (e.g imported software libraries or hardware drivers) is bigger than the probability of fingerprint collisions. Nevertheless, we support 3 modes for the store operation to fulfill the requirements of applications that may need absolute certainty that fake duplicates do not occur: *force-new*, *regular* and *compare*. When using the force-new mode, the elimination of duplicates is switched off and a new block is created to store each document. This semantic is useful if one knows that the collection does not contain duplicates. The *regular* mode (default) detects a collision if two contents have the same signature but different sizes. In this case, an overflow block is created to keep the document. However, the success of this heuristic depends on the distribution of the document sizes and collisions will not be detected among documents with the same size. We computed the distribution of sizes for a random sample of 3.2 million web pages and found that the probability of two random web pages having the most frequent size is 4.15 x $10^{-6}$. Assuming that the probability of two pages having the same size and the probability of fingerprint collision between them are independent events, our results indicate that the comparison of sizes can substantially reduce the occurrence of fake duplicates without requiring longer fingerprint signatures or additional meta-data. The *compare* mode relies on size and bytewise comparison of the documents to detect collisions. If two contents have the same signature but different sizes, or have the same size but are not byte equal, a collision is detected. Fake duplicates never occur when using this store mode.

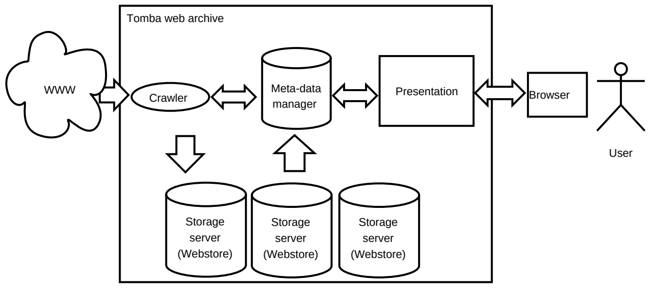
Tomba web archive
WWW
Crawler
Meta-data manager
Presentation
Browser
User
Storage server (Webstore)
Storage server (Webstore)
Storage server (Webstore)

**Figure 2: Overview of the Tomba archive.**

## 5. WEBSTORE

### 5.1 The Tomba web archive

The Tomba web archive periodically crawls and archives textual documents from the Portuguese web. Figure 2 presents an overview of the system. The *crawlers* harvest the web looking for documents belonging to the Portuguese web and store them in *storage servers* spread across several hosts. The reference to the physical location of each document is stored in the *meta-data manager* [8], along with other meta-data such as URL, type of document or date of crawl. Users request the archived versions of a given URL through a web interface. The *presentation* component finds the physical locations of the archived documents in the meta-data manager and retrieves the documents from the storage servers. Finally, the documents are presented to the user on the browser. A web archive has high performance requirements because it must be intensively loaded with new documents gathered from the web at the same time it provides concurrent access to the archived ones. The storage servers were implemented using the Network File System (NFS) [7]. It seemed an attractive idea because NFS is widely available, free and well tested. However, it imposes changes on the underlying operating system and administrative privileges on clients and servers to be installed. This became a problem when we had to run crawlers on machines that were not under our administration. Besides, the crawlers had been unexpectedly loosing documents and suffering from crashes of the Java Virtual Machine while they were storing documents through NFS. As the web archive system evolved we witnessed that NFS could not cope with high loads such as the ones inflicted by the crawler. This experience was also witnessed by other researchers. The Dominos crawler de-

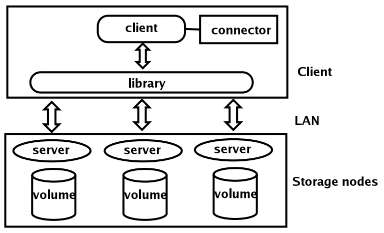

**Figure 3: Architecture of the system.**

veloped to facilitate French legal deposit used NFS to manage the collected documents but the authors identified efficiency problems and are planning on migrating to a different platform [21]. Shkapenyuk and Suel designed their crawler to store and compress the downloaded pages on a NFS based repository and bottlenecks had also been detected with this component [37]. Ana Patterson also alerted to problems arose from using NFS on a search engine [30]. Thus, there was a need for an adequate storage manager able to cope with the requirements of web warehousing applications. This need motivated the development of the Webstore prototype, an easily manageable, performant storage manager.

## 5.2 Prototype

Webstore was developed to validate the duplicates elimination mechanism and as replacement of the Tomba storage servers. It presents a distributed architecture composed by set of autonomous nodes that provide disk space with no central point of coordination, extending storage capacity by adding new nodes without imposing major changes in the system. Although network file systems also provide distributed access to data, they are executed at operating system kernel level and require administrative privileges to be installed and operated [34]. On its turn, Webstore's software is platform independent and runs at application level without imposing changes in the configuration of the underlying operating system. Peer-to-peer file systems, such as Oceanstore [33], are designed to manage a large and highly variable set of nodes with small storage capacity, distributed over wide-area networks (typically the Internet). This raises specific problems and imposes complex intra-node communication protocols that guarantee properties such as security, anonymity or fault tolerance that unnecessarily limit throughput on controlled networks. An experiment showed that Oceanstore is on average 8.3 times slower than NFS on a LAN. Webstore has a different scope, because it assumes nodes with large storage capacity located within the same local-area network and rare changes on the set of storage nodes. The collections managed by Webstore contain millions of documents with identical probabilities of being read; therefore, caching mechanisms are not required. We assume that all the nodes have similar distance costs to the clients and there are no malicious clients. The system tolerates faults of storage nodes, but it does not provide automatic recovery and full availability of the documents kept in the faulty nodes. However, Webstore provides methods that enable the implementation of replication policies within an instance.

Figure 3 depicts the architecture of Webstore. An instance is composed by a thin middleware *library*, the *connector* object and the *volume* servers. Clients access an instance through a connector object that keeps references to the volumes that compose the instance. A change in the composition of the instance, such as the addition of a new volume, implies an update of the connector. The clients execute operations through the invocation of methods provided by the library's API. Each volume server manages the requests and executes the correspondent low-level operations to access the documents. The documents are transmitted between the library and the servers in a compressed format to reduce network traffic and data processing on the server.

The storage structure of a volume is implemented as a directory tree over the filesystem where the blocks are files residing at the leaf directories. The block header is written in ASCII format so that it can be easily interpreted, enabling access to the document kept in the block independently from the Webstore software. We used a 64-bit implementation of Rabin's fingerprinting algorithm to generate document signatures [32]. Webstore supports Zlib as the built-in compression method but other compression algorithms can be dynamically loaded. This way, documents can be compressed using adequate algorithms and accommodate new formats. The connector object was implemented as an XML file. The library and the volume servers were written in Java using JDK 1.4.2. The communication between them is through Berkeley sockets. Currently, clients access volume servers in sequence, (a multicast protocol is not yet implemented). Volume servers are multi-threaded, launching a thread for handling each request. Volume servers guarantee that each block is accessed in exclusive mode through internal block access lists.

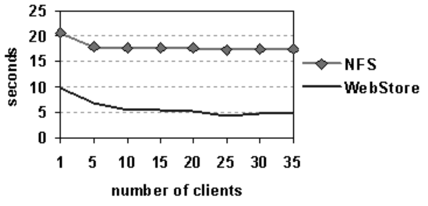

**Figure 4: NFS read vs. Webstore retrieve.**

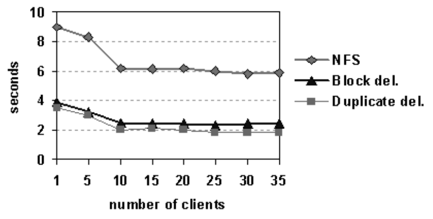

**Figure 5: NFS remove vs. Webstore the delete.**

## 6. EXPERIMENTS

In this Section, we present the results gathered from a set of 4 experiments ran on Webstore and NFS. We used NFS on Linux (nfs-utils-1.0.1-2.9) as the baseline because it is widely known and accessible, enabling the reproducibility of

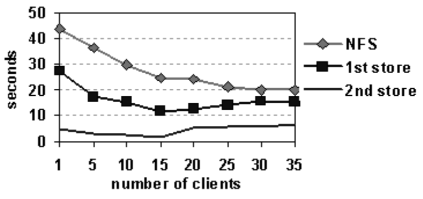

**Figure 6: NFS save vs. Webstore regular store.**

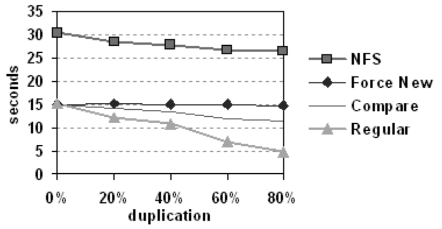

**Figure 7: Time spent to store the data set presenting increasing levels of duplication.**

our experiments. Plus, we wanted to evaluate Webstore as replacement of the Tomba storage servers which were implemented using NFS. We used Pentium 4 at 2.4 GHz machines with Red Hat Linux 9.0 to run our experiments. The disks are IDE ATA 100, 7200 r.p.m. and data is managed by a software RAID 5 controller. We provide a detailed description of the hardware used on a extended report [19]. We developed clients in Java for each one of the experiments. The Webstore clients executed the operations through the invocation of methods available in the API library and the NFS clients used the JDK IO library to execute equivalent operations on a remote directory mounted on the local file system. The data set used in the experiments was composed by 1000 distinct HTML pages with a total size of 19.2 MB gathered from a previous crawl of the Portuguese web.

**Retrieving:** In this experiment, we loaded the Webstore volume server with the pages from the data set, compress them using Zlib, and kept all the generated contentkeys. Then, we split the contentkeys among the clients and ordered them to retrieve the corresponding documents. In each measurement, we added a new machine hosting 5 clients to stress the servers. For the NFS test we followed an equivalent procedure: we stored the files, kept the path and ordered the clients to read them in parallel. Before each measurement, we re-mounted the NFS volumes to prevent caching of the files. Figure 4 presents the total time that the clients took to read all the documents from Webstore and NFS. We found that Webstore is on average 68% faster than NFS for read operations. The total time for executing the task in Webstore remained constant for more than 10 parallel clients with Webstore and more than 5 clients with NFS.

**Deleting:** We loaded the data set into the Webstore and NFS volumes and split the references to the documents among the clients, as described in the previous experiment. We launched the clients that deleted the documents from the data set. Then we loaded the data set into Webstore twice, creating a situation where all the documents were duplicates and relaunched the clients to delete them. Figure 5 compares the total time that the clients took to delete all the documents from NFS and Webstore. Our system is on average 67% faster than NFS when deleting a duplicate, (only the reference counter is decremented in this case), and 60% faster than NFS when deleting the block. We believe that Webstore outperforms NFS because it uses a lighter protocol of communication and does not have the overhead of maintaining caches of files and associated meta-data such as permissions. Both the NFS and Webstore servers reached their maximum throughput with 10 clients.

**Storing:** In this experiment, we split the data set among the clients that stored it in Webstore and NFS. For each set of clients, we stored the data set twice in Webstore to compare the times spent creating blocks to store new documents and add references in the presence of duplicates. The NFS volume was exported with the *sync* option, which does not allow the server to reply to requests before the changes made by the request are written to the disk (same behaviour as Webstore). The experiments on NFS had to be restarted several times due to crashes of the Java Virtual Machine that were running the clients. We could not determine exactly the cause of this problem neither reproduce the errors but, as this situation never occurred with the Webstore clients, this strengthen our suspicion that the crawlers were crashing due to an incompatibility between the JDK IO library and NFS. Figure 6 presents the total times spent storing the documents in Webstore and NFS. As expected, the store operation is faster for storing duplicates than for storing new documents, because the documents are not transferred from the clients to the volume servers and the creation of new blocks is not required. For duplicated documents Webstore outperformed NFS by 82%. For new documents it outperformed NFS on average by 50%, from 5 to 20 clients. However, it seems to reach its saturation point with 15 clients, while NFS achieves it only with 25 clients. We found that as Webstore is faster than NFS, it reached the server's disk throughput peak performance earlier as we rose the number of clients.

**Semantics of the store operation:** The objective of this experiment was to measure the performance of the 3 different modes available for the store operation: regular, compare and force-new. We gradually increased by 20% the level of duplication within the data set and launched 10 clients that stored the data set using each one of the modes. Figure 7 presents the results obtained. The semantic of the force-new mode is similar to the NFS write operation, given that it does not put any effort in eliminating duplicates. However, it took almost half of the time to finish the task. As the level of duplication increased, the compare and regular modes presented the best results. The compare mode is slower than the regular, because it avoids fake duplicates through bytewise comparison, while the regular mode compares only the sizes of the contents. When there was not replication within the data set, all the three modes performed the same way. Therefore, we conclude that the overhead of detecting duplicates within a volume is insignificant. Besides saving disk space, this experiment shows that the proposed mechanism for the elimination of duplicates increases the storage throughput of the system when it manages collections containing duplicates.

## 7. RELATED WORK

The Internet Archive is building a digital library of Internet sites and other cultural artifacts in digital form. It keeps documents and associated meta-data in large aggregate files (100MB) in the ARC format. These files are stored in a conventional file system [6].

We found several web archive initiatives that claimed to have implemented their own storage system for web documents but do not provide in-depth descriptions of the system: the PANDORA web archive developed their own Digital Object Storage System [10], a project that aims to archive the greek web used an SQL server database and XML files [26] and the the Swedish Kulturarw3 project used a commercial hierarchical storage management system [31].

The NEDLIB project had the purpose of developing harvesting software specifically for the collection of Web resources for a European deposit library [22]. The meta-data was kept in a relational MySQL database while the crawled documents were stored in a unix file system either on disk or tape. The documents were assembled in unique files compressed using the ZIP algorithm. Each document is identified by a MD5 checksum kept in a central database, which is used to detect and eliminate duplicates within the archive. Daily, a process called the *Archiver* consults the MD5 database to detect duplicates and transfers the documents crawled to a different directory. We could not found a performance evaluation of this strategy but we believe that for large data sets the archive operation would cause a significant disk IO overhead on the host machine that would delay the crawler or other applications accessing the data stored in it. This storage approach differs from the one we followed in Webstore mainly because it is based on centralized storage, imposes the same compression method regardless the documents format and executes off-line elimination of duplicates.

The Mercator crawler executes a content-seen test that identifies duplicate pages, to prevent multiple processing of the same document and reduce the number of pages to be downloaded. This mechanism relies on two independent sets of fingerprints: a small hash table kept in memory and a large sorted list kept in a single disk file [23].

Honicky and Miller presented a fast algorithm to allocate data objects to disks. It enables efficient load balancing and data distribution among a heterogeneous cluster of servers through the usage of weighting. The algorithm allows to adjust the degree of replication of each object [24]. However, it does not eliminate duplicates among objects with distinct identifiers. Nevertheless, an implementation of a system presenting the features described would be useful to rapidly store and locate documents within a web archive.

Crespo and Garcia-Molina proposed a multi-layer architecture formed by a federation of independent, heterogeneous but collaborating sites, that used signatures as object handles. The main purpose of this architecture was to ensure the preservation documents and relationships among them, through a replication schema [14]. However, the proposed architecture assumes that there are not voluntarily deletions, which may be necessary to dispose uninteresting documents crawled from the web such as error messages or pages generated by spider traps [23]. Later, this architecture was used in the Stanford Archival Repository Project where a simulators were included to evaluate the effectiveness of an archive over a long time span [13].

The Webase project studied how to construct and keep a large shared repository of web pages fresh [12]. The research paper of the popular search engine Google described a repository where web pages are stored sequentially in compressed packets [4]. Herodotus is an web archival system based on a Peer-to-Peer architecture [5].

## 8. CONCLUSIONS AND FUTURE WORK

We presented a study of the persistence of URLs and documents within a web archive over two years. We could not found a similar study, so we believe this is an original contribution. Based on the obtained results and related work we discussed the elimination of partial and exact duplicates within a web archive. We concluded that exact duplicates are frequent within a web archive and URLs are highly transient, so storage techniques based on the persistence of document identifiers are not adequate. We believe that the elimination of partial duplicates is not adequate for web archives because it raises preservation issues and can not be efficiently applied in large distributed storage systems. So, we proposed an algorithm to eliminate exact duplicates during a crawl of the web. We implemented the algorithm for eliminating duplicates in the Webstore storage manager. The algorithm for eliminating duplicates, besides saving space, increases storage throughput. The experimental results showed the storage manager using duplicates elimination outperformed significantly NFS in read, write and delete operations. Webstore is platform-independent and runs at the application level. The source code of Webstore and the clients used to gather the presented results are available for download at `http://webstore.sourceforge.net/`.

Webstore has been intensively tested in several usage contexts. It was used to store bioinformatics research papers, document corpora in cross evaluation initiatives [9] and the last crawls performed for a Portuguese search engine [38]. Our experience using Webstore showed that it is easily manageable, lightweight and flexible. We concluded that it can be easily integrated in existing systems. Webstore was included as a component of the Tomba web archive and it currently hosts a total of 57 million web documents (1.3 TB). These documents were migrated from previous crawls hosted in NFS and directly loaded by the crawler in recent harvests. The beta version of the Tomba Portuguese web archive is available to the public at `http://tomba.tumba.pt`.

We are currently studying a larger web data set to derive document characteristics that may influence the persistence of URLs and documents. We intend to derive mathematical models for estimating the lifetime of URLs and unchanged documents. Our experiences using Webstore enabled us to detect possible optimizations. The development of an application-level multicast protocol to efficiently eliminate duplicates across multiple volumes is planned for future work.

## 9. REFERENCES

[1] B. Berliner. CVS II: Parallelizing software development. In *Proceedings of the USENIX Winter 1990 Technical Conference*, pages 341–352, Berkeley, CA, 1990. USENIX Association.

[2] K. Bharat and A. Broder. Mirror, mirror on the web: a study of host pairs with replicated content. In *Proceedings of the Eighth International Conference on World Wide Web*, pages 1579–1590. Elsevier North-Holland, Inc., 1999.

[3] S. Brin, J. Davis, and H. García-Molina. Copy detection mechanisms for digital documents. pages 398–409, 1995.

[4] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, 1998.

[5] T. Burkard. Herodotus: A peer-to-peer web archival system, 2002.

[6] M. Burner and B. Kahle. WWW Archive File Format Specification, September 1996.

[7] B. Callaghan, B. Pawlowski, and P. Staubach. *RFC 1813: NFS Version 3 Protocol Specification*. Sun Microsystems, Inc., June 1995.

[8] J. Campos. Versus: a web repository. Master thesis, 2003.

[9] N. Cardoso, M. J. Silva, and M. Costa. The XLDB Group at CLEF'2004.

[10] W. Cathro and T. Boston. Development of a digital services architecture at the national library of australia. *EduCause, Australasia 2003*, page 24, 2003.

[11] J. Cho and H. Garcia-Molina. Estimating frequency of change. *ACM Trans. Inter. Tech.*, 3(3):256–290, 2003.

[12] J. Cho, H. Garcia-Molina, T. Haveliwala, W. Lam, A. Paepcke, S. Raghavan, and G. Wesley. Stanford webbase components and applications. Technical report, Stanford Database Group, July 2004.

[13] B. F. Cooper, A. Crespo, and H. Garcia-Molina. The stanford archival repository project: Preserving our digital past. Technical report 2002-47, Department of Computer Science, Stanford University, October 2002.

[14] A. Crespo and H. Garcia-Molina. Archival storage for digital libraries. In *Proceedings of the Third ACM International Conference on Digital Libraries*, 1998.

[15] L. Daigle, D. van Gulik, R. Iannella, and P. Faltstrom. *Uniform Resource Names (URN) Namespace Definition Mechanisms*, October 2002.

[16] T. Denehy and W. Hsu. Duplicate management for reference data. Technical report RJ 10305, IBM Research, October 2003.

[17] D. Fetterly, M. Manasse, M. Najork, and J. Wiener. A large-scale study of the evolution of web pages. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*, pages 669–678, New York, NY, USA, 2003. ACM Press.

[18] R. A. Finkel, A. Zaslavsky, K. Monostori, and H. Schmidt. Signature extraction for overlap detection in documents. In M. J. Oudshoorn, editor, *Twenty-Fifth Australasian Computer Science Conference (ACSC2002)*, Melbourne, Australia, 2002. ACS.

[19] D. Gomes, A. L. Santos, and M. J. Silva. Webstore: A manager for incremental storage of contents. DI/FCUL TR 04–15, Department of Informatics, University of Lisbon, November 2004.

[20] D. Gomes and M. J. Silva. Characterizing a national community web. *ACM Trans. Inter. Tech.*, 5(3):508–531, 2005.

[21] Y. Hafri and C. Djeraba. Dominos: A new web crawler's design. In *4th International Web Archiving Workshop (IWAW04)*, Bath, UK, September 2004.

[22] J. Hakala. Collecting and preserving the web: Developing and testing the nedlib harvester. *RLG Diginews*, 5(2), April 2001.

[23] A. Heydon and M. Najork. Mercator: A scalable, extensible web crawler. *World Wide Web*, 2(4):219–229, 1999.

[24] R. J. Honicky and E. L. Miller. A fast algorithm for online placement and reorganization of replicated data. Nice, France, Apr. 2003.

[25] T. Kelly and J. Mogul. Aliasing on the world wide web: Prevalence and performance implications. In *Proceedings of the 11th International World Wide Web Conference*, Honolulu, Hawaii, May 2002.

[26] C. Lampos, M. Eirinaki, D. Jevtuchova, and M. Vazirgiannis. Archiving the greek web. In *4th International Web Archiving Workshop (IWAW04)*, Bath, UK, September 2004.

[27] J. MacDonald. Versioned file archiving, compression, and distribution.

[28] J. Mogul. A trace-based analysis of duplicate suppression in HTTP. Technical Report 99/2, Compaq Computer Corporation Western Research Laboratory, November 1999.

[29] N. Noronha, J. P. Campos, D. Gomes, M. J. Silva, and J. Borbinha. A deposit for digital collections. In *Proc. 5td European Conf. Research and Advanced Technology for Digital Libraries, ECDL*, pages 200–212. Springer-Verlag, 2001.

[30] A. Patterson. Why writing your own search engine is hard. *Queue*, 2(2):48–53, 2004.

[31] K. Persson. Kulturarw description. http://www.kb.se/kw3/ENG/Description.htm, March 2005.

[32] M. O. Rabin. Probabilistic algorithm in finite fields. Technical Report MIT/LCS/TR-213, 1979.

[33] S. Rhea, P. Eaton, D. Geels, H. Weatherspoon, B. Zhao, and J. Kubiatowicz. Pond: the Oceanstore Prototype. In *Proceedings of the 2nd Usenix Conference on File and Storage Technologies (FAST'03)*, 2003.

[34] O. Rodeh and A. Teperman. zfs: A scalable distributed file system using object disks. In *Proceedings of the 20 th IEEE/11 th NASA Goddard Conference on Mass Storage Systems and Technologies (MSS'03)*, page 207. IEEE Computer Society, 2003. zFS extends the research done in the DSF project.

[35] N. Shivakumar and H. García-Molina. SCAM: A copy detection mechanism for digital documents. In *Proceedings of the Second Annual Conference on the Theory and Practice of Digital Libraries*, 1995.

[36] N. Shivakumar and H. Garcia-Molina. Finding near-replicas of documents and servers on the web. In *Selected papers from the International Workshop on The World Wide Web and Databases*, pages 204–212. Springer-Verlag, 1999.

[37] V. Shkapenyuk and T. Suel. Design and implementation of a high-performance distributed web crawler. In *Proceedings of the 18th International Conference on Data Engineering (ICDE'02)*, page 357. IEEE Computer Society, 2002.

[38] M. J. Silva. Searching and archiving the web with tumba! In *CAPSI 2003 - 4a. Conferência da Associação Portuguesa de Sistemas de Informação*, Porto, Portugal, November 2003.

[39] D. Spinellis. The decay and failures of web references. *Communications of the ACM*, 46(1):71–77, 2003.

[40] L. L. You and C. Karamanolis. Evaluation of efficient archival storage techniques. In *21st IEEE / 12th NASA Goddard Conference on Mass Storage Systems and Technologies*, College Park, MD, April 2004.