# Implementation of a high performance architecture for managing and storing web-harvested collections

***Michael Smorul, Joseph JaJa; Institute for Advanced Computer Studies, Department of Electrical and Computer Engineering, University of Maryland; College Park, Maryland***

## Abstract

*As institutions continue to grow their collections of web-harvested content, there is an ever increasing need for tools that organize, index and share this data. Even a modest web crawl consisting of a few web sites may generate millions of harvested documents. Repeating these crawls over time greatly expands the complexity of stored data. Identifying the scope of a crawl, the location of a page within a crawl and the differences over time between crawls becomes a challenging task. In this paper we will describe a software architecture in use at the University of Maryland designed to support research on quickly extracting information about the crawls, including statistical information, and on indexing web content. While designed to support research, many of the challenges addressed in this software exist at any site which has to manage large sets of time-spanning data.*

*Our architecture consists of two components. The first is a database application for organizing WARC-based web data called a WarcManager. The WarcManager was designed to track URL location and to allow easy extraction of crawl statistics across collections of warc-stored data. It provides both a REST-based API to harvested data as well as a portal for viewing statistics across the collection. The second component is a high performance, http based, storage service called the Simple Web-Accessible Preservation(SWAP) system. The SWAP system is distributed, novel file placement and retrieval service. It has been designed to be minimally intrusive and to allow complete data recovery even in the absence of any SWAP software.*

*These two components have been used to successfully support research into high performance indexing of web-based content. We will describe the implementation and performance characteristics of each component as well as possible real-world uses for the system.*

## Introduction

The ADAPT Project at the University of Maryland is currently researching problems surrounding temporal text searching of web archives. During the course of this research UMD has been allowed to borrow several large web crawl collections from the Library of Congress. These collections consist of data from many sites, crawled repeatedly over the period of several months. Immediately after receiving this data, several questions arose.

1. What data do we have?
2. What are the characteristics of the crawls?
3. How can subsets of the data be extracted?
4. How can the data be effectively stored and distributed to a high performance compute cluster for further exploration?

The two web collections we received contained 660GB and 8.6TB of unstructured data respectively. The storage to host these collections was only available in 3.4TB partitions. Compounding the problem, raid arrays were connected to multiple servers and each array could stream data at a maximum sustained rate of 60MB/s, or about 50% of the total bandwidth for each server. To effectively distribute data to a high performance compute cluster, data would need to be served from multiple disk arrays, stored on multiple servers.

Answering these challenges resulted in the development of two pieces of software. First the Warc Manager was created to provide a database of all items in the test collections. The Warc manager is a database application which stores metadata about items in ARC[1] and WARC[2] files. The database stores enough metadata to allow researchers to quickly determine how many copies of a URL exist in an archive, where those copies are located, and how many unique versions or duplicates exist within the archive.

The second component, the Simple Web-Accessible Preservation (SWAP) system was developed to provide a lightweight storage solution for managing terabytes of data. In order to overcome the disk limitations in the lab, the system had to be able to split data across multiple partitions stored on several servers. To accommodate access by high performance clusters, careful consideration had to be given to avoid introducing bottlenecks. To solve both of these tasks, SWAP emerged as a set of servers able to intelligently place data without requiring a central file location catalog.

We will now describe the Warc Manager and its functionality followed by a description of the SWAP approach to data storage.

## Warc Manager

The Warc Manager is a web application which consists of two parts. The first is a MySQL database used to store metadata about each page in a web archive. The second piece is a REST-API[5] used to provide programmatic access for ingesting new contend and querying existing archives. The Warc Manager was built using common open source components including Tomcat, MySQL, and Java.

The MySQL database captures the following information about each page in a collection:

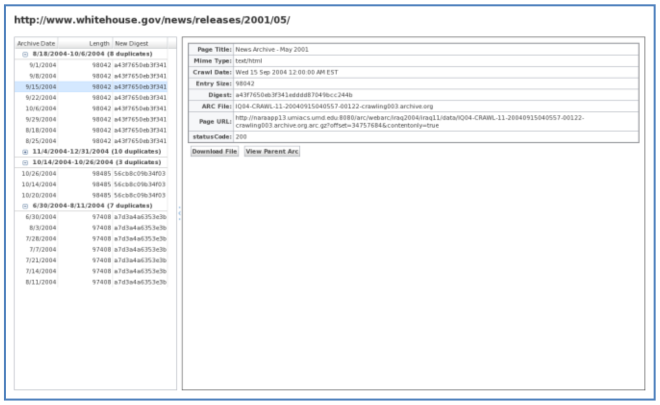
- Offset and physical location in a ARC/WARC file.
- URL of each page with a full-text index
- Metadata about the page including digest, crawl date, and mime type.
- Container contents for each WARC/ARC file in the collection

On top of this database, a REST-API was developed for data ingest and querying.

### REST-API

The REST-API provides programmatic to the underlying collection data base for querying and ingesting of new content. Specifically, the API provides the following functionality:

- Register new ARC/WARC files
- Query for URL's based on a free-text string
- Retrieve the detailed contents of any WARC/ARC file in the collection.
- List all ARC files in a collection.
- Retrieve details for a URL including metadata and crawl/duplicate entry information.



http://www.whitehouse.gov/news/releases/2001/05/

| Archive Date | Length | New Digest |
|---|---|---|
| 8/18/2004-10/6/2004 (8 duplicates) | | |
| 9/1/2004 | 98042 | a43f7650eb3f341 |
| 9/8/2004 | 98042 | a43f7650eb3f341 |
| 9/15/2004 | 98042 | a43f7650eb3f341 |
| 9/22/2004 | 98042 | a43f7650eb3f341 |
| 10/6/2004 | 98042 | a43f7650eb3f341 |
| 9/29/2004 | 98042 | a43f7650eb3f341 |
| 8/18/2004 | 98042 | a43f7650eb3f341 |
| 8/25/2004 | 98042 | a43f7650eb3f341 |
| 11/4/2004-12/31/2004 (10 duplicates) | | |
| 10/14/2004-10/26/2004 (3 duplicates) | | |
| 10/26/2004 | 98485 | 56cb8c09b34f03 |
| 10/14/2004 | 98485 | 56cb8c09b34f03 |
| 10/20/2004 | 98485 | 56cb8c09b34f03 |
| 6/30/2004-8/11/2004 (7 duplicates) | | |
| 6/30/2004 | 97408 | a7d3a4a6353e3b |
| 8/3/2004 | 97408 | a7d3a4a6353e3b |
| 7/28/2004 | 97408 | a7d3a4a6353e3b |
| 7/7/2004 | 97408 | a7d3a4a6353e3b |
| 7/21/2004 | 97408 | a7d3a4a6353e3b |
| 7/14/2004 | 97408 | a7d3a4a6353e3b |
| 8/11/2004 | 97408 | a7d3a4a6353e3b |

| | |
|---|---|
| Page Title: | News Archive - May 2001 |
| Mime Type: | text/html |
| Crawl Date: | Wed 15 Sep 2004 12:00:00 AM EST |
| Entry Size: | 98042 |
| Digest: | a43f7650eb3f341edddd87049bcc244b |
| ARC File: | IQ04-CRAWL-11-20040915040557-00122-crawling003.archive.org |
| Page URL: | http://naraapp13.umiacs.umd.edu:8080/arc/webarc/iraq2004/iraq11/data/IQ04-CRAWL-11-20040915040557-00122-crawling003.archive.org.arc.gz?offset=34757684&contentonly=true |
| statusCode: | 200 |

***Figure 5: Detailed URL information***

Layered above this REST-API a javascript interface eas developed to provide user-friendly access to collections. This interface permits users to browse the contents of a web archive, search groups of URL's and discover multiple versions of pages. Users may specify search patterns to extract all the pages in a particular site. Figure 5 shows a summary for a single URL in our collection. This summary shows multiple versions of a page, multiple crawl dates, and metadata for each instance when the page was crawled.

### Production Installation

The Warc Manager fully indexes the two collections provided by the Library of Congress. A script was developed to read each ARC file and extract the information required by the Warc Manager. The collection has the following characteristics.

- 177 million unique entries
- 37.4 million unique URL's
- 63,694 ARC files occupying 9.2Tb of storage

### Next Steps

The Warc Manager provides a useful portal into large archives of web content. It allows any individual to quickly browse the ARC files in a collection, list what URLS have been crawled and determine if duplicates exist for any URL. Beyond answering these basic questions a few more requirements have arisen which will be addresses in future versions. Specifically some of the questions we anticipate being able to answer about collections are:

- For any object in an archive, what links point to it?
- What possible paths exist in a collection from one URL to another?
- How temporally complete is a page? How close in age are all the contents of a given page?

## SWAP Approach

The SWAP approach is a simple solution to data placement which provides an http-based API for accessing data distributed across multiple disk partitions and servers. It was designed to be completely disposable, meaning that no data will be lost even in the event all SWAP software is removed from the data servers.

### Dividing a namespace

In SWAP, data is organized into generic collections. Collections are defined to be data that is somehow related and the size of a collection may vary from a few gigabytes to many terabytes containing millions of files. Each of the web crawls we received was considered one collection. Collections are split up into a number n of slices. Each slice is then assigned to a different disk partition. Each slice is assigned a unique index between 0 and n -1.

Data in a collection is split up and written to one of the slices. To determine which slice should receive the data, we calculate the result of the MD5 digest of the file path relative within the collection modulo the number of slices in the collection.

Slice s = MD5(path) % number of slices

The slice resulting from this calculation is responsible for storing the data for the given path. To locate any file in a collection you only need to know the locations of all the slices in a collection, and their index values. This removes the need for a file location catalog. Additionally, if a server tracks the locations of all slices in a collection, it is able to return the location of any file to a client in the event it receives a request for a file that does not reside on one of its slices.

### Managing Namespace

A primary goal of SWAP was to allow collections to be unified by recursively copying the data in all component slices into a single directory. This allows recovery of a collection without any requirement on the SWAP software. To allow for this recovery, a few simple rules on how a path is represented are required. First, a path separator of '/' is always used regardless of platform, second all slices must reside on the same type of filesystem. Limitations on characters, etc are not imposed, but non-standard characters may limit rebuilding of collections to the type of filesystem in which the slices currently reside.

While files will map to a specific slice, directories will exist across slices. For example, assume the file 'documents/myfile1.txt' maps to slice 1 and documents/myfile2.txt maps to slice 2. This results in the directory 'documents' existing on both slice 1 and slice 2.

In order to write data into SWAP, a check must be performed to guard against namespace conflicts. A conflict may occur when attempting to write a file onto a slice where part of the path is a physical file on another slice. For example, a file is uploaded with the path ‘documents/project1’. Assume the placement algorithm determined this file was to be placed on slice 1. Later, another request is made to upload a file with the path ‘documents/project1/readme.txt’. Assume the placement algorithm maps this file to slice 2. If this file is written to slice 2, a collision will exist on the ‘project1’ component of the path. Attempts to merge the slices into a single collection will fail.

To solve this problem, whenever a new directory is encountered in a path, a slice must check with its peer slices to ensure no conflict exists before creating the path. Failure to do so will result in a namespace conflict and an inability to recombine the collection and create a unified version. Through caching and checking pre-existing directories performance concerns can be mitigated.

### *Scaling*

We are able to efficiently scale a collection across many slices by doubling the number of slices in a collection. After doubling, data must be rebalanced on to the new slices. On each slice, only approximately 50% of all data will require moving to the new slices, the other half will continue to remain on the original slice. Let’s examine a two slice collection which doubles to 4 slices. The doubling process will result in approximately half the data from slice 0 migrating to slice 2 and half data from slice 1 migrating to slice 3. No data will movement will occur between the original slices 0 and 1. Each file path on each slice will have to be processed through the placement algorithm to determine which data will move.

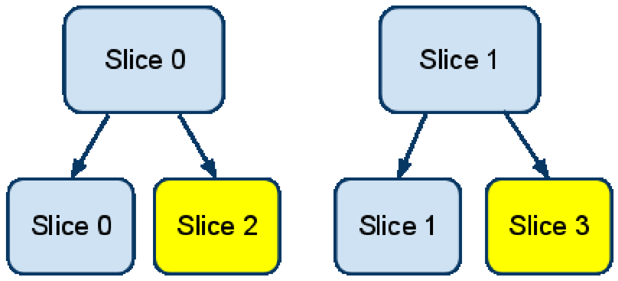


***Figure 1: Rebalancing a collection***

By reversing the process, we reduce the number of slices to by cutting the number of slices, in half and moving all data from the upper slices s to their corresponding lower peer. If an attempt is made to increase or decrease the number of slices in a collection, the amount of data movement will increase significantly as data will likely not be able to remain on the old nodes.

Using these two techniques, we can scale a collection up to dozens of slices and all the way down to one slice.

### *Collection Organization*

Collections within a SWAP network are combined into a unified namespace. Any file in SWAP has a unique path which consists of two parts. The first is a collection namespace and the second is the local namespace within a collection. The collection namespace helps define the collection’s address within a hierarchy. While each collection has a unique namespace, parts of that namespace my overlap with other collections. For example the Chronopolis collections at UMIACS all share a starting prefix of ‘chronopolis’ in their collection name. Collections from individual data suppliers share a prefix containing ‘chronopolis’ and the provider name. For example the umiacs data provider has two collections, code and documentation. These collections would appear in SWAP as: ‘chronopolis/umiacs/code’ and chronopolis/umiacs/documentation’.

## Architecture and Implementation

SWAP has been implemented as a standalone server which distributes data using http and communicates with a set of trusted peers to manage data placement and lookup. The server consists of two parts, first is an http/1.1[4] compatible server which clients and other servers use for file ingest and access. The second part is a tcp socket connection between peers and management software to locate slices of a collection.

### *HTTP Server*

Each SWAP server runs a webserver which distributes data to multiple clients. This webserver presents a global URL hierarchy based on the unified namespace of the SWAP network. For example, to access a file with relative collection path ‘documents/project1/readme.txt’ and a collection with namespace ‘chronopolis/umiacs/code’ residing on a server ‘swapserver1.umiacs.umd.edu’ a client would connect to the URL ‘http://swapserver1.umiacs.umd.edu:8080/get/chronopolis/umiacs/code/documents/project1/readme.txt’. As each SWAP server tracks the locations of all slices in the network, servers are able to return an http 302 redirect to clients which request files from the wrong server. This allows clients to know the identity of a single SWAP server and still be able to retrieve any data stored in the network. This redirect is illustrated in figure 2

1. A Client send an HTTP GET request to Server-1 for the file ‘documents/file’ in a collection.
2. Server-1 executes the placement algorithm and determines the file resides on slice 3 which it does not host
3. Server-1 returns an HTTP 302/Redirect to the client with a URL that contains the server responsible for slice 3.
4. The client issues another HTTP GET request to the correct server.
5. The server returns either the requested file, or a 404/not found response to the client.

In addition to distributing data using http, SWAP also uses http to receive data. To upload data into SWAP, a client issues an HTTP PUT request containing the file to be uploaded to any server. If it contacts the wrong server, it will receive a redirect response pointing to the server which will store the data.

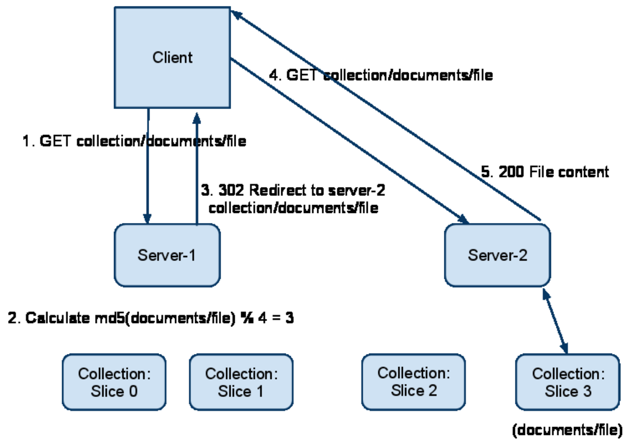

Client
4. GET collection/documents/file
1. GET collection/documents/file
5. 200 File content
3. 302 Redirect to server-2 collection/documents/file
Server-1
Server-2
2. Calculate md5(documents/file) % 4 = 3
Collection: Slice 0
Collection: Slice 1
Collection: Slice 2
Collection: Slice 3
(documents/file)

***Figure 2: HTTP Request flow***

An advantage of using primitive HTTP PUT and GET requests is the wide variety of tools available which can understand the protocol. The unix command-line tool 'curl' is able to follow 302 redirects can be used to upload files to a SWAP server. For example, the following command will upload the file 'screenshot.png' into swap:

```
curl –L –T screenshot.png http://server:8080/put/collection/screenshot.png
```

Expanding this example, we can combine 'curl' with the 'find' utility to upload unlimited amounts of data into SWAP. At UMIACS, this method has successfully been used to move dozens of terabytes from local disk into SWAP resources.

```
find . –type f –exec curl –L –T http://server:8080/put/collection/documents/{} \;
```

In addition to command-line utilizes, any web browser is able to retrieve data out of a SWAP system.

## *Server Architecture*

A SWAP server organizes slices into a set of partitions, each of these partitions may contain at most one slice from any one collection. Partitions store both data on disk organized by its collection and local namespace. This disk layout was designed to allow for easy access to SWAP-managed data outside of the SWAP environment. For example, the file 'documents/project1/readme.txt' stored in the 'chronopolis/umiacs/code' collection written to a partition located at '/disk1/swap-partition' would have an absolute unix path of '/disk1/swap-partition/chronopolis/umiacs/code/documents/project1/readme.txt'. Partitions contain a simple metadata file in its root directory which contains a unique identifier for the partition and a brief description.

Each SWAP server also opens a TCP socket which is used to coordinate slice location and provide management functions. In order to effectively redirect and accept new data, SWAP servers maintain a persistent connection to all peers. When a server is brought online, it immediately attempts to contact its peers. In addition to contacting peers upon startup, a background task periodically checks an internal peer list and attempts to reestablish communication with any offline server. When a connection is established to a peer server, a complete list of collections and slices are exchanged. These slice and collection lists are used to direct clients to the correct SWAP server when a request for a remote file is received.

In order to handle the directory/file collision described above, this management connection is used to query remote servers to ensure a collision isn't possible. When checking to see if a collision occurs, a server analyzes the parent path of a file, and will recursively check each that path until it is determined the partial path is a directory, or a collision is detected. Servers will cache the results of previous peer queries to avoid excessive inter-node messages. The steps a server follows to determine if a directory contains no namespace collision follows:

1. Check locally to see if directory for path exists in slice partition.
2. Check cache to see if peer responded true.
3. Query peer responsible for path.
4. Repeat check using parent path unless collection root has been reached.

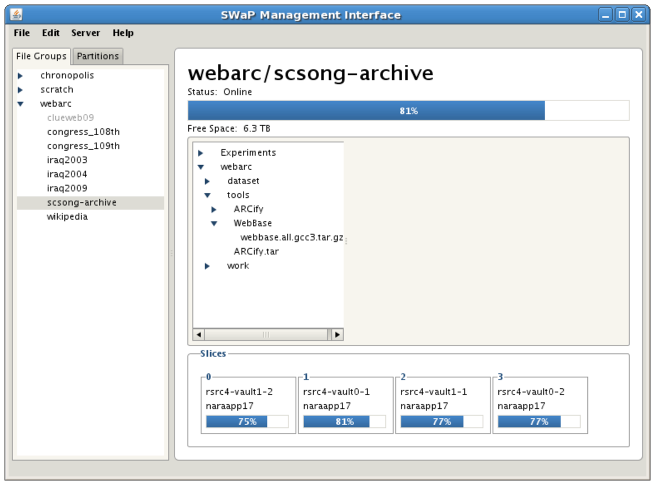


***Figure 3: SWAP Management Interface***

This management network also provides an API to query file attributes, create new collections, remove data, and list server status. A management user interface(UI) has been built to allow archive administrators to easily create, modify, and delete collections and files in a SWAP network. This management UI is shown in figure 3.

## *Performance and Use-Cases*

A set of SWAP servers has been deployed at the University of Maryland to manage the storage for two projects. The first project is the web indexing project described earlier. The second use is to manage storage for the UMIACS Chronopolis Digial Preservation Project site. Data for these two projects are hosted on a set of Apple XRAIDS connected to multiple dual-core Sun AMD servers. These servers each had a 1Gbps network interface. Local performance testing demonstrated the peak performance of a single raid partition was approximately 50-60MB/s, or 50% of the available bandwidth of a server.

By deploying the SWAP system across 8 partitions divided between two servers, we were able demonstrate a sustained transfer approaching 1Gbps over several hours. This performance approached the peak capacity of the network link between the SWAP storage nodes and the high performance compute cluster used for indexing. Figure 4 shows this traffic.

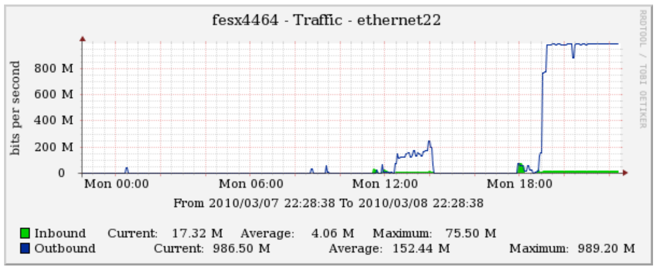


***Figure 4: Network traffic graph for SWAP node uplink***

Furthermore, the two servers were able to distribute small files (4k) at a rate in excess of 2000 files per second per node and handle redirects for missed requests at over 3000 requests per second.

## Conclusion

Through the development of the Warc Manager and the Simple Web-Accessible Preservation system we have created a software stack which has demonstrated a method for storing and managing large collections of web content. Furthermore, the SWAP system has shown how to distribute data in an easy to manage, yet fully recoverable fashion. Both of these tools are still under active development and will likely be released under an open source license in the future.

## References

[1] Mike Burner, Brewster Kahle, ARC File Format, http://www.archive.org/web/researcher/ArcFileFormat.php

[2] ISO 28500:2009, Information and documentation -- WARC file format ISO/DIS 28500

[3] The MD5 Message-Digest Algorithm http://tools.ietf.org/html/rfc1321

[4] Hypertext Transfer Protocol -- HTTP/1.1 http://tools.ietf.org/html/rfc2616

[5] Fielding, Roy T.; Taylor, Richard N. (2002-05), "Principled Design of the Modern Web Architecture" (PDF), ACM Transactions on Internet Technology (TOIT) (New York: Association for Computing Machinery) 2 (2): 115–150, doi:10.1145/514183.514185, ISSN 1533-5399

## Author Biography

*Mike Smorul is currently lead programmer for the UMIACS ADAPT Project at the University of Maryland, College Park. He oversees development of the Audit Control Environment, and tool and storage design for the Chronopolis preservation environment. He received his BS in computer science from the University of Maryland, College Park.*

*Joseph JaJa currently holds the position of Professor of Electrical and Computer Engineering with a joint appointment at the Institute for Advanced Computer Studies at the University of Maryland, College Park. Dr. JaJa received his Ph.D. degree in Applied Mathematics from Harvard University and has since published extensively in a number of areas including parallel and distributed computing, combinatorial optimization, algebraic complexity, VLSI architectures, and data-intensive computing. His current research interests are in parallel algorithms, digital preservation, and scientific visualization of large scale data. Dr. JaJa has received numerous awards including the IEEE Fellow Award in 1996, the 1997 R&D Award for the development software for tuning parallel programs, and the ACM Fellow Award in 2000. He served on several editorial boards, and is currently serving as a subject area editor for the Journal of Parallel and Distributed Computing and as an editor for the International Journal of Foundations of Computer Science.*