Recovering a Website's Server Components from the Web Infrastructure

Frank McCown
Harding University
Computer Science Department
Searcy, Arkansas, USA 72149
fmccown@harding.edu

Michael L. Nelson
Old Dominion University
Computer Science Department
Norfolk, Virginia, USA 23529
mln@cs.odu.edu

ABSTRACT

Our previous research has shown that the collective behavior of search engine caches (e.g., Google, Yahoo, Live Search) and web archives (e.g., Internet Archive) results in the uncoordinated but large-scale refreshing and migrating of web resources. Interacting with these caches and archives, which we call the Web Infrastructure (WI), allows entire websites to be reconstructed in an approach we call *lazy preservation*. Unfortunately, the WI only captures the client-side view of a web resource. While this may be useful for recovering much of the content of a website, it is not helpful for restoring the scripts, web server configuration, databases, and other server-side components responsible for the construction of the website's resources.

This paper proposes a novel technique for storing and recovering the server-side components of a website from the WI. Using erasure codes to embed the server-side components as HTML comments throughout the website, we can effectively reconstruct all the server components of a website when only a portion of the client-side resources have been extracted from the WI. We present the results of a preliminary study that baselines the lazy preservation of ten EPrints repositories and then examines the preservation of an EPrints repository that uses the erasure code technique to store the server-side EPrints software throughout the website. We found nearly 100% of the EPrints components were recoverable from the WI just two weeks after the repository came online, and it remained recoverable four months after it was "lost".

Categories and Subject Descriptors: H.3.5 [Information Storage and Retrieval] Online Information Services: Web-based services; H.3.7 [Information Storage and Retrieval] Digital Libraries: Collection

General Terms: Design, Experimentation, Measurement

Keywords: backup, digital preservation, search engine caches, web archiving, web server

JCDL'08, June 16–20, 2008, Pittsburgh, Pennsylvania, USA.

1. INTRODUCTION

As the most popular publishing medium today, the Web has unleashed exponential growth of mankind's creative output, and along with it, an abundance of curatorial and preservation challenges. Individuals do not recognize their personal responsibility for preserving personal data like websites: a 2006 survey indicates only 57% of individuals who store personal data on their computers ever backup their data [5], and even experts who work on backup storage techniques admit they do not regularly backup their personal files [7]. Webmasters who rely on ISPs and web hosting companies to preserve their websites are often surprised and disappointed to discover that these organizations are not immune to viruses, hackers, bankruptcy and run-ins with the law [9, 13, 18]. Even if a great deal of personal care and energy are expended to properly ensure the availability of the individual's or organization's website, the website may still become defunct once it is no longer needed, financial pressures dictate ending the website or the maintainer dies [13]. In this case, interested third parties may have few means available to resurrect the website since they typically do not have access to backups.

In response to the ephemeral nature of the Web, we have created Warrick, a web-repository crawler which can reconstruct lost websites from search engine caches and web archives (collectively known as the Web Infrastructure or WI) [22]. Warrick is currently being used by the public to reconstruct over 100 websites a month using Google, Live Search, Yahoo and the Internet Archive (IA) repositories [21]. Some individuals use Warrick to recover their own lost websites, and others use it to recover websites that once belonged to third parties.

In a survey we conducted of Warrick users [13, 18], some have lamented that the functionality behind their dynamic websites was not recoverable. Recovering the content is often useful, but recovering the scripts, databases, and other servers-side components would be ideal for restoring the functionality of the lost website. Search engines and web archives generally use web crawlers to fill their repositories, and because server components are not accessible to web crawlers, this content cannot be recovered from the WI.

In this paper, we review methods that could be used to recover server-side components from the WI and perform a proof-of-concept experiment with a novel method inspired by steganography, where information is hidden within a larger context [11]. Our method uses erasure codes which have been used in a variety of applications like RAID systems [25], secret-sharing [10] and information dispersal [26].

Erasure codes are used to divide a message into blocks where the original message can be recovered from a subset of those blocks [10]. We split the server components into many pieces using erasure codes, then inject those pieces into crawlable portions of our website in an unobtrusive way, thus using the WI as a back-up system when they crawl and cache/achive our website. Even when only a small number of pages can be recovered from the WI, the entire set of server components is recoverable. In our experiment, we installed an EPrints repository and obserbed that nearly 100% of the server components were recoverable from the WI just two weeks after the repository came online; the components remained recoverable three months after the repository was "lost".

2. BACKGROUND AND RELATED WORK

Web archiving projects have attempted to capture historic snapshots of the Web for posterity and research purposes, and search engines often make cached copies of web pages available in case of transient errors. Both web archives and search engines primarily rely on web crawling to fill their repositories. Web crawling is a long-standing method for creating a snap-shot of a website's generated content, but it is often limited to pages that are not protected from the robots exclusion protocol [12] and pages that remain hidden in the deep web [4].

As pointed out by Masanès [14], some archiving projects like the Internet Archive and Sweden's Kulturarw3 [2] have focused on crawling a wide range of websites using primarily automated methods. Other archives have used more labor-intensive processes to guarantee completeness. For example, the France's BnF archive augments crawling with "legal deposit"; archive staff works directly with webmasters to deposit their websites within the archive so 'deep web' sites will remain functional in the future [1]. Search engines have also made strides in indexing deep web content [17], but we are unaware of any effort to cache web server components.

In previous work [16, 22], we have shown that the WI can be used effectively to recover a large number of websites if they were lost today. The WI provides *lazy preservation*: large-coverage preservation for the Web with no effort required of the content producer. All types of resources (html, PDF, images, etc.) are accessible from the WI if they become inaccessible from the Web. Recent experiments have shown that if a typical website were to become lost with no backup, 61% of its resources (77% textual, 42% images and 32% other) could be recovered from the WI using Warrick [16]. However, these recoveries are unable to reproduce the server components of the website.

We have identified three methods that could be used to inject server components into the WI:

1. **Exposing the raw components** - The server components of a website can be combined into one or more compressed files which are then placed on the web server and exposed to the WI.

2. **Robot vaults** - The server components can be encoded into special crawlable pages which are created solely for WI crawlers.

3. **Dispersion through preexisting content** - The server components can be injected into preexisting crawlable pages in an unobtrusive manner.

The first method is the simplest, but because search engines do not prefer compressed binary content [22], it is likely to only be captured by web archives. This leaves the components particularly vulnerable to loss.

The second method has been explored by Traeger et al. [29]. They demonstrated how search engine caches could be used for backing-up a filesystem by creating special HTML pages designed only for storage; their pages did not contain readable information that would be useful to the public at large. While all members of the WI are likely to crawl these pages, search engines might consider them spam and would therefore not store them in their caches. Additionally, if a single page was not cached, the embedded file was lost.

We propose the final method which uses previously existing pages of a website for storing the server components and thereby voids the risk of creating spam. Taking advantage of the fact that HTML allows comments of any size to be inserted within the page without affecting the appearance of the page, this approach can hide the encoding from the viewer of the page. Like steganography, the encodings are hidden from view. Additionally, erasure codes can be used to spread the encodings to multiple pages where recovery of only a subset of the pages allows complete recovery of the server component. This mitigates the risk incurred by the robot vaults method of requiring a single page to be stored by the WI in order to recover a single server component.

3. INJECTION MECHANICS

3.1 What to Protect

When considering which server components to protect, two approaches can be used. Using a *pessimistic approach*, the worst possible scenario is planned for, and as many server components are protected as possible: scripts, database contents, and crawlable content like images, style sheets, and PDFs that may not be stored in a canonical format by all members of the WI. Other resources could be included like the script interpreter, third party libraries, database software, and even the operating system if there is a possibility that any of these service components are in danger of being lost.

Using a more *optimistic approach*, protection is given to as few resources as possible without losing the functionality of the website. In this case only the customized scripts and database contents are protected, and it is assumed the WI would store any crawlable content in its canonical format, and all third-party supporting software would be readily available in an emergency. The optimistic approach puts a much lower storage burden on the WI while increasing the risk of losing certain components.

The pessimistic approach is essentially computing the transitive closure on software dependencies necessary for long-term preservation. The tension between the optimistic and pessimistic approaches is one that frequently occurs in digital preservation projects, such as documenting popular formats and encodings in [30] and the knowledge base and representation information of OAIS [6].

3.2 Dispersing Encoded Components

The server components of a website can be injected into crawlable pages by base64 encoding the components and inserting the encodings into HTML comments. The comments

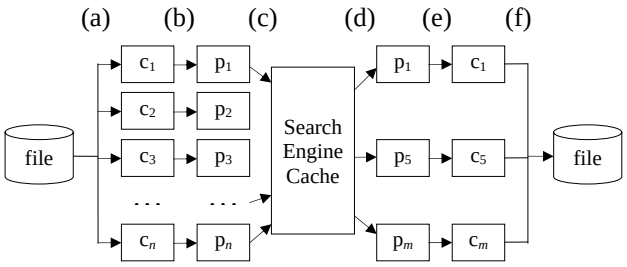


Figure 1: Injecting and recovering a server file into and from the WI.

can be injected into any pre-existing page on the target website.

An erasure code transforms a message of l blocks into a message with n blocks (where $l < n$) where the original message can be recovered from a subset of those blocks [10]. Figure 1 illustrates how erasure codes can be used to inject a server file into crawlable content:

a) Use erasure codes to break an optionally encrypted server file into n blocks where recovery of any r blocks allows for complete recovery of the encrypted file.

b) Insert each block (base64 encoded) and metadata into n HTML files.

c) Wait for search engines (or other web repositories) to crawl and cache the HTML files.

Once a sufficient number of pages have been cached or archived, the server file can be recovered like so:

d) Recover as many HTML files from search engine caches as possible (m).

e) Extract available blocks and metadata from the recovered HTML files.

f) If at least r blocks have been recovered (where $r \leq m$), reconstruct the server file (and optionally decrypt) using the erasure codes.

3.3 Segmenting Approaches

Since dynamically generated websites normally contain a large number of server files (sometimes much larger than the number of indexable pages), it is best to compress the server files together into one or more zip files. This reduces the amount of data injected into the WI and is practically easier to manage. When choosing how to segment the server files into zip files, two different approaches can be taken:

1) **All-or-nothing approach** – Compress all server files together into a single file and inject it into all available pages.

2) **Segmented approach** – Compress groups of server files into single files based on some sort of grouping mechanism (e.g., by directory, change rate, last modified, importance, etc.) and inject them into subsets of available pages.

The first approach is the simplest to implement, but if the minimum number of blocks are not recovered, then *none* of the server components are recovered. Also, if one server file is changed, the single compressed file must be recreated and re-inserted into all the HTML pages. A web repository would then need to re-crawl and re-store a large number of pages it had already cached or archived in order to have stored all the blocks of the same version.

Although more complicated, the second approach allows recovery of individual groups of server components when a minimum number of blocks are recovered. Also, if one server file is changed, only the compressed file containing the changed server file must be recreated and re-inserted, thus isolating the changes to a subset of HTML files.

Figure 2 illustrates these two approaches. In the all-or-nothing approach (left), all the server files are compressed together into a single zip file, and the file's blocks are then distributed to all the available pages. With the segmented approach (right), groups of server files are grouped together into separate zip files which are then distributed to subsets of pages. Larger zip files (in bytes) are allocated a larger number of pages. For each subset, r pages would need to be recovered (where r is specific to the subset) to reconstruct the zipped server file.

When a website is lost, complete knowledge of where each zipped file was injected and which zipped files are missing is ideal. For example, if none of the pages could be found in the WI that house the first zip file, knowledge that the file is missing (and knowledge of its contents) can help the recoverer when restoring the website's functionality. A **manifest** listing the filenames, paths, timestamp, sizes and details about where the components were injected can be produced and distributed along with the zipped server components. Inserting the manifest into every recoverable page would ensure its recoverability at the cost of re-building and re-injecting the manifest every time a file changed. To minimize these costs, the manifest could be inserted into one or more pages that have a high likelihood of being recovered, like the root page of the site or the pages a single hop away from the root page.

3.4 Choosing Block Sizes

When choosing values of n and r, care must be taken to ensure the block sizes are not too large since search engines truncate cached resources that are beyond their respective thresholds (Google: 977 KB, Yahoo: 214 KB, Live: 1 MB) [20]. The block size b can be calculated using the size of the zipped server file s like so:

$$b = s/r \tag{1}$$

As r increases, the size of the blocks will decrease, but the minimum number of HTML files to store r blocks increases. Therefore, a minimum value for r should be used which takes into account the search engine's cache size limit t, the maximum block size or load z that should be added to the website's pages, the total number of HTML resources h whose size is $< t - z$, and the size of the zipped server file s. The minimum value of r for a zipped file is calculated like so:

$$r = \lceil s/z \rceil \tag{2}$$

The z parameter is the only one which can be easily manipulated. Ideally a value of z should be picked that is as small as possible so the resulting pages are not overly large,

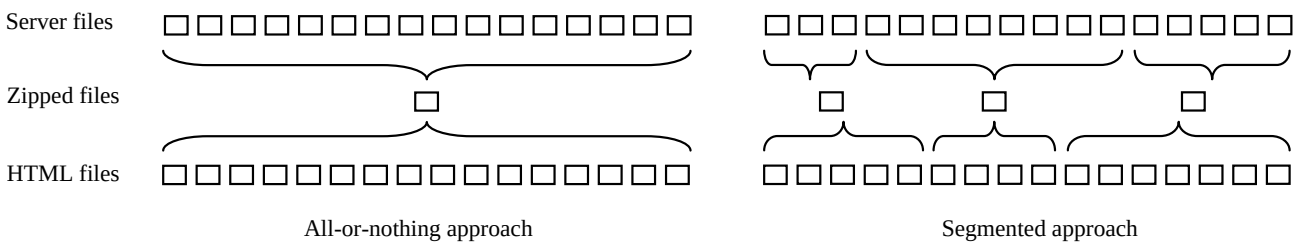


Figure 2: Approaches to injecting server components into web pages.

Table 1: Various r values (bold).

Zipped file size s	Block size z (before base64 encoding) 50 KB	100 KB	150 KB
1 KB	**1**	**1**	**1**
1 MB	**21**	**11**	**7**
100 MB	**2,048**	**1,024**	**683**
1 GB	**20,972**	**10,486**	**6,991**

causing them to download slower and creating more of a burden for the repository. Consideration should also be made for the extra space taken by base64 encoding since base64 encoding expands the block size by one third. If $r > h$ then there are not enough HTML pages to store the minimum number of blocks r. In this case the acceptable block size (z) would need to be increased, or if possible, more pages could be added to the website (thus increasing h) or the size of the server file (s) could be decreased by removing less essential server components. Once r has been calculated, n blocks can be created where $n = h$.

To illustrate how the size of the zipped file s and the block size limit z affects the number of HTML pages required to store the blocks, the value of r has been calculated for various values of s and z in Table 1. Since the average size of HTML files varies between 10-40 KB [3, 24] and since Yahoo is the most restrictive repository with only 214 KB of cache space, blocks should generally be no larger than 130 KB (solve for x where 214 KB - 40 KB = $1.33x$).

3.5 Versioning

When web server components change or new ones are added, the injected pages must be updated with newly computed blocks. Additionally, metadata about each block needs to be kept along with the blocks for version control since only blocks that contain the same version of a server file can be used together to reconstruct the server file.

Since search engines often crawl the same pages only once a day or less often to be polite to the web server, blocks can be re-computed on a daily basis, ideally at times when server activity is low. Alternately, a web server module could be used to inject pages on-the-fly with encoded blocks depending on the identity (IP address or user agent) of the requester. This would allow smaller pages to be served to regular users and larger pages with the encoded blocks to be served to search engines. This technique of serving clients different pages based on their identity is called *cloaking*, and it must be used with caution since most search engines disapprove of it [31].

3.6 Security Considerations

The WI is accessible to everyone; this has its benefits and its drawbacks. If a website owner was to die and their site became lost over time, the site's users may want to recover the site so it would remain accessible to the site's user community. Having the server components readily accessible from the WI would dramatically decrease the effort involved in making the site functional once again.

On the other hand, the site may contain sensitive data like passwords, account IDs, credit card numbers, etc. which the site owner may *never* want accessible to a third party. In this case the server components could be encrypted with a private key, and death of the site owner (and knowledge of the key) would likely prevent the functionality of the site from ever being recovered (at least until the encryption was broken). The site owner should also consider what would happen if the key were ever forgotten or compromised.

In a third scenario, the website owner may want to protect their server components in the immediate future but not necessarily for all time. Use of non-parallelizable cryptography, i.e. timed-release crypto [28], could be used to keep the server components protected until a set amount of time had passed.

4. EXPERIMENTS

To validate the feasibility of recovering a website's server components from the WI, two experiments were conducted using websites that dynamically produce a majority of their content. The experiments were limited to digital repositories which were running GNU EPrints [8], a popular open source repository package that is composed of Perl scripts, configuration files and MySQL database. Ten randomly selected repositories running EPrints were crawled and reconstructed to see how much content might be recovered for a typical EPrints repository. The server components for these repositories were not recoverable since they did not implement any injection methods. In the second experiment, a test repository was created, the Monarch Repository, using EPrints software. It contained 100 PDF resources and embedded server encodings. The repository was reconstructed on a weekly basis for 32 weeks.

4.1 Reconstructing 10 Digital Repositories

The first experiment examined how much of an EPrints repository could be recovered from the WI if the repository was to suddenly disappear. Ten randomly selected repositories running EPrints were selected from the Registry of Open Access Repositories (ROAR) [27]. The same methodology from previous reconstruction experiments were used to de-

termine reconstruction success[19, 16, 22]: the repositories were crawled using the Heritrix web crawler [23] and then reconstructed with Warrick, and the reconstructions were compared to the crawled sites. Of course the reconstructions were only able to recover the client-side representation of the sites and none of the server components.

A distinguishing characteristic about repositories is that they typically curate a large number of non-HTML resources like historical images, academic documents, and the like. They maintain a variety of metadata about each resource for purposes of provenance. For many repositories, PDF documents are the primary resources being curated, and much of the metadata can be re-generated as long as the PDF remains available. The ten repositories selected in this experiment contained a variety of resource formats, but they all contained a number of PDFs.

Table 2 lists the ten repositories in order of total resources (HTML, images, PDFs, etc.) along with the difference vectors and reconstruction diagrams (difference vectors are the percent of resources changed, missing and added, and reconstruction diagrams are visualizations of difference vectors as introduced in [22] and explained at the bottom of the table). On average, 87% of the repository's resources were recovered and 83% of the PDFs. Unfortunately, the PDFs recovered were much more likely to be in HTML format rather than their native format (68% vs. 32%) since search engines convert PDFs into HTML documents when cached and IA had few PDFs archived. This conversion may be acceptable for PDFs that are purely textual, but loss of figures and other images in the PDF-to-HTML conversion process are usually problematic in the face of complete loss.

4.2 Recovering a Repo's Server Components

4.2.1 Setup

The second experiment was a proof-of-concept, demonstrating the erasure code injection method presented previously could be used to recover the server components of a website. A digital repository was created using EPrints and populated with 100 academic papers in PDF format and metadata (the papers were from the fields of Web technologies and information retrieval). The PDFs were all previously accessible on the Web. Since the PDFs were accessible to a web crawler, it is possible that a majority of them would duplicate PDFs already cached by the search engines. However, Google Scholar frequently provides many alternate locations of the same PDF and maintains multiple cached copies. Although Warrick does not pull directly from Google Scholar, we found the same PDFs indexed by Google Scholar were always accessible from the Google cache used by Warrick.

An example web page from the Monarch Repository (as it was called) is shown in Figure 3. If the user places the cursor over the PDF icon, a preview image of the document appears. This PNG image can easily be found by a web crawler. Although each page notified the reader that it was from a "test repository," we decided it was unlikely a search engine would refuse to cache the page because of the presence of that phrase. The composition of the repository as seen by a typical web crawler is shown in Table 3.

The optimistic approach was adopted to preserve the server components: only the Eprints software (Perl scripts), configuration files and database contents (extracted with the mysqldump tool) were preserved. There was no private or sensitive data in the EPrints server components. The size of the software was approximately 3.3 MB (uncompressed), 2 MB for config and other miscellaneous files and 650 KB for the database contents. Tarring and compressing all these files together with gzip produced a 1 MB file. The PDFs occupied 41 MB of space and would have added 30 MB to the compressed tar file.

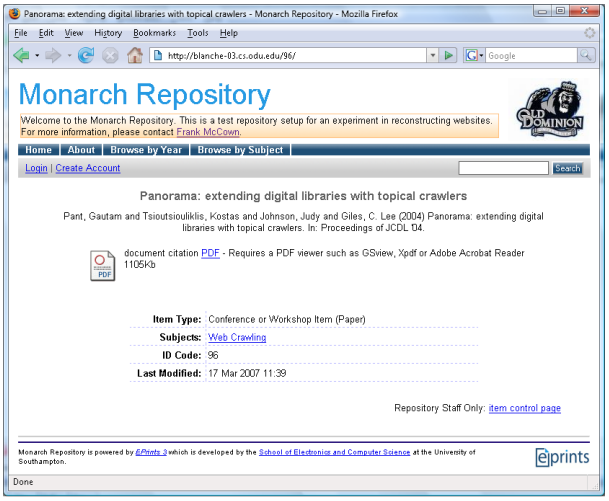


Figure 3: Monarch Repository screen shot.

Table 3: Composition of Monarch Repository.

Type	Total	Distribution
HTML	123	34.0%
Images	110	30.4%
PDF	100	27.6%
Style sheet	26	7.2%
Other	3	0.8%
Total	362	100%

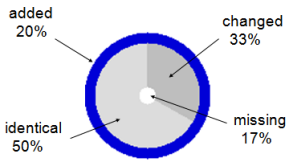
The encoded server components were injected into the HTML resources of the website. An example of an injected block in HTML comments is shown in Figure 4.

For 19 weeks, the Monarch Repository was crawled with Heritrix and reconstructed with Warrick (using the Comprehensive policy which attempts to recover a maximum of all lost resources [19]) at the end of the week as was performed in the previous reconstruction experiment. The crawls were matched with the reconstructions to produce an accurate assessment as to how much of the website was being successfully reconstructed each week. After week 19, the site was taken off-line to simulate "losing" the website, and only reconstructions were performed weekly.

Several techniques were used throughout the experiment to test the all-or-nothing and segmented approaches and website updates. Initially, the segmented approach was used to create ten compressed tar files, one for each directory of the Eprints software and the database (s ranged from 122 bytes to 462 KB). The files were dispersed among the 123 HTML pages according to size ($h = 123$ and $z = 50$ KB), so larger files were allocated more pages than smaller files which kept the block sizes from becoming too large. The

Table 2: Ten reconstructed digital repositories.

Repository	All resources		PDFs				Diff	Recon
	Total	Recov	Total	Recov	as PDF	as HTML	vector	diag
1. eprints.libr.port.ac.uk	176	97.2%	36	94.4%	2.9%	97.1%	(0.722, 0.028, 0.018)	
2. archiviomarini.sp.unipi.it	222	86.0%	47	46.8%	13.6%	86.4%	(0.658, 0.131, 0.000)	
3. eprints.erpanet.org	272	82.0%	66	60.6%	75.0%	25.0%	(0.467, 0.062, 0.229)	
4. open.ekduniya.net	452	95.4%	87	81.6%	4.2%	95.8%	(0.699, 0.046, 0.005)	
5. brief.weburb.dk	458	88.4%	143	82.5%	72.0%	28.0%	(0.286, 0.109, 0.022)	
6. eprints.bbk.ac.uk	771	93.4%	331	91.8%	1.3%	98.7%	(0.720, 0.065, 0.004)	
7. eprints.vu.edu.au	1192	98.1%	314	96.5%	84.2%	15.8%	(0.316, 0.017, 0.004)	
8. eprints.lse.ac.uk	1336	95.0%	513	89.5%	2.8%	97.2%	(0.821, 0.046, 0.003)	
9. www.cbmh.ca	1695	99.6%	673	99.6%	12.2%	87.8%	(0.602, 0.004, 0.003)	
10. bnarchives.yorku.ca	2130	30.3%	272	84.6%	47.4%	52.6%	(0.173, 0.690, 0.012)	
Average	870.4	86.5%	248.2	82.8%	31.6%	68.4%	(0.546, 0.120, 0.030)	



Difference vector (changed, missing, added)

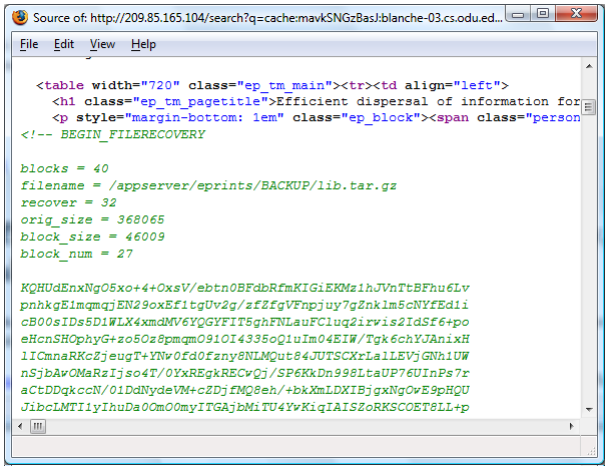


Figure 4: Encoding of a server file in HTML comments from the Monarch Repository.

Table 4: Updates and changes made throughout the experiment.

Week	Summary
1	Ten segments are distributed throughout the website.
10	Single link is added to unlinked web page.
13	Subset of blocks are reallocated to eight web pages.
16	All server components are reallocated to all web pages.
19	Website is taken off-line.

manifest was encoded and placed in the root page of the repository. The resulting pages averaged approximately 60 MB in size, far below Yahoo's 214 KB size limit. On May 19, 2007, links pointing to the repository were placed on three previously indexed pages on the `www.cs.odu.edu` website in order to advertise the existence of the website to the WI.

After nine weeks, a single link was added to the root page which pointed to a single web page which had accidentally been unlinked. This unlinked page contained one of the encoded blocks which was needed to recover one of the ten server files. Three weeks later, several of the encoded blocks were re-arranged and re-allocated to eight web pages to simulate an update to the website. And three weeks later, the all-or-nothing approach was tested by creating a single gzipped tar file for all the repository server components and distributing it among the entire website. A "This page was modified on *date and time*" notice was also added to each page in this last phase to encourage the search engines to re-cache all the pages.

Finally, the repository was taken off-line on week 19 to simulate the loss of the website. The website was configured to return an HTTP 404 (not found) response to every URL request except for the root page which contained a notice that the website had been taken off-line. These steps are summarized in Table 4.

4.2.2 *Results*

Figure 5 shows how much of the Monarch Repository was recovered each week. The 362 resources are ordered on the

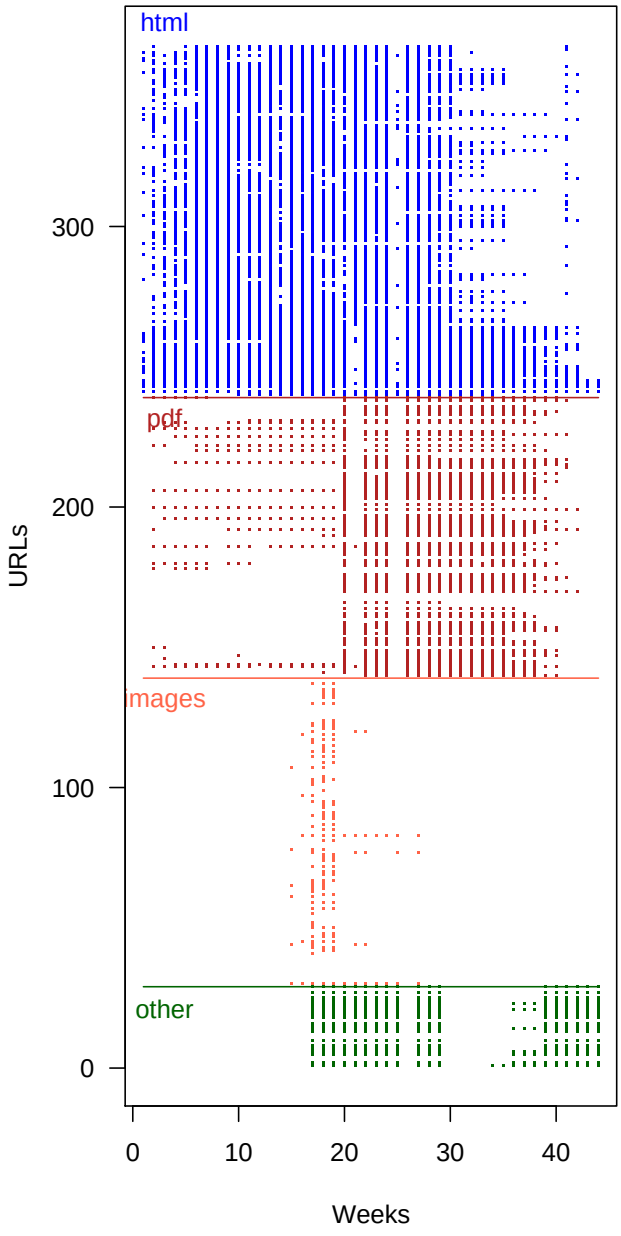


Figure 5: Recovered Monarch resources each week. Each dot represents the successful recovery of the resource on the given week.

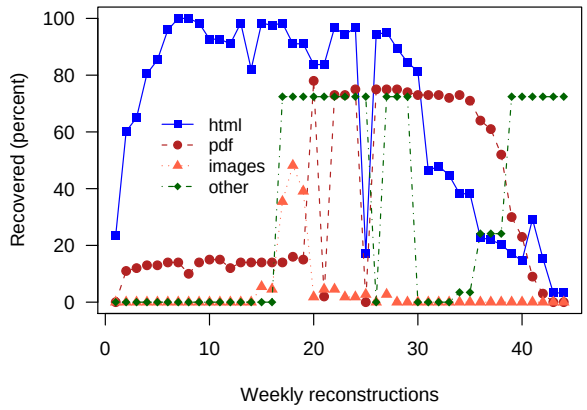


Figure 6: Percent of recovered Monarch resources each week.

y-axis by resource type. A square dot indicates that the resource was recovered that week. The percentage of resources recovered each week is plotted in Figure 6. Just a few days after a link was created to the Monarch Repository, Google discovered and crawled the website, making a quarter of the discovered pages available from their cache. Each week the percentage of resources recovered from Google increased. Live and Yahoo had crawled a number of resources from the repository a few weeks after Google, but neither search engine made anything available from their cache until later in the experiment as will be discussed shortly. Several style sheets (categorized as "other") were recovered from IA beginning on week 17, only two months after they first crawled the website.

On week 7 and 8, 100% of the HTML resources were recovered (minus one unlinked page), all from Google. However, on week 9 several HTML resources which were once accessible from Google's cache were no longer cached, and a 100% recovery rate for HTML resources was not observed again. As illustrated by Figure 5, a small number of URLs tended to fluctuate in and out of Google's cache throughout the experiment. For example, the URL `http://blanche-03.cs.odu.edu/118/` was cached on weeks 1, 2, 4–9, 13–17.

Although Google crawled a large number of images on week 7, only a handful of images were recovered from Google by week 15; interestingly, one of them appeared to be a blank image (Figure 7). Live had cached 18 images by week 15, but Warrick was not able to recover them due to a bug in the Live API [15]. Unlike the high percentage of recovered PDFs from the first experiment (Table 2), Figure 6 shows only a small percentage of the Monarch's PDFs were made available from Google's cache; none were accessible from Live and Yahoo.

The server components were far more recoverable than the resources making up the client view. Table 5 lists the percentage of recovered HTML pages and server files each week and the contribution rate of each repository throughout the experiment. Some of the weeks indicate an update to the website was performed prior to that week's reconstruction. Almost 100% of the server components were recoverable from the WI just two weeks after the experiment began, despite having only recovered 60% of the HTML pages. It was not until week 10, when a link was posted to an un-

Table 5: Recovered server files and repository contributions each week.

Week	Recov HTML (%)	Recov server files (%)	Contributors (%)			
			Google	Live	Yahoo	IA
1	23.6	59.4	100	0	0	0
2	60.2	99.6	100	0	0	0
3	65.0	94.6	98.9	1.1	0	0
4	80.5	99.7	100	0	0	0
5	85.4	99.7	99.2	0.8	0	0
6	95.9	99.7	99.2	0.8	0	0
7	100	99.7	97.1	2.9	0	0
8	100	99.7	98.5	1.5	0	0
9	98.4	99.7	98.5	1.5	0	0
10[1]	92.7	100	100	0	0	0
11	92.7	100	88.4	11.6	0	0
12	91.1	100	89.5	10.5	0	0
13[2]	98.4	92.1	85.2	14.8	0	0
14	82.1	92.1	69.6	17.4	13.0	0
15	98.4	100	81.6	14.2	4.2	0
16[3]	97.6	0	80.6	11.5	7.9	0
17	98.4	100	68.7	8.2	12.3	10.8
18	91.1	100	66.3	6.9	16.3	10.4
19[4]	91.1	100	47.1	5.8	36.1	11.0
20	83.7	100	2.5	66.2	21.2	10.3
21	83.7	100	9.9	52.7	21.4	16.0
22	96.7	100	3.7	79.4	7.3	9.6
23	94.3	100	3.8	83.5	2.8	9.9
24	96.7	100	4.1	82.9	3.2	9.7
25	17.1	0	44.4	0	6.7	48.9
26	94.3	100	1.0	97.9	1.0	0
27	95.1	100	2.8	86.6	0.9	9.7
28	89.4	100	0	89.3	0.5	10.2
29	84.6	100	1.5	87.4	0.5	10.6
30	81.3	100	0	100	0	0
31	46.3	100	1.5	98.5	0	0
32	48.0	100	0.8	99.2	0	0
33	44.7	100	1.6	98.4	0	0
34	38.2	100	0	99.2	0	0.8
35	38.2	100	0	99.2	0	0.8
36	22.8	100	1.0	92.0	0	7.1
37	22.0	0	0	91.6	0	8.4
38	20.3	0	0	90.5	0	9.5
39	17.1	0	1.4	68.1	0	30.6
40	14.6	0	1.6	62.9	0	35.5
41	29.3	100	1.5	66.7	0	31.8
42	15.4	0	2.3	46.5	0	51.2
43	3.3	0	4.0	0	0	96.0
44	3.3	0	4.0	0	0	96.0

[1]Single link is added to unlinked web page.
[2]Subset of blocks are reallocated to eight web pages.
[3]All server components are reallocated to all web pages.
[4]Website is taken off-line.

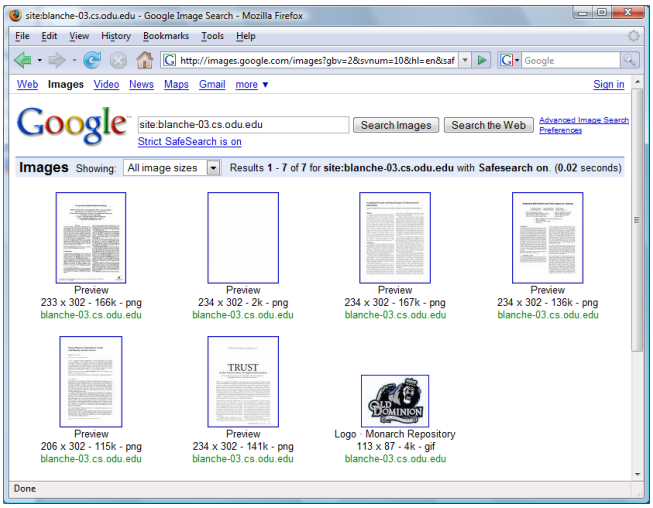


Figure 7: Images from Monarch Repository cached by Google.

linked web page, that all 100% of the server files could be recovered.

When several encoded blocks were reallocated to eight web pages at the beginning of week 13, the percent of recoverable server files dropped accordingly. The web server logs indicated that Google, Live and Yahoo continued to crawl the affected pages after the changes were made, but the cached pages were not being updated. Since the vast majority of pages were dynamically generated and did not return a Last-Modified HTTP header, the search engines may have performed some type of processing on the crawled pages which ignored changes to HTML comments only. It took nearly three weeks before three of the eight pages were recoverable which allowed all 100% of the server files to be recovered once again.

By the end of week 16, none of the server components were recoverable. This was due to the complete reallocation of all the server components at the beginning of the week which required the altered pages to be re-crawled and cached. Since the WI seemed to mostly ignore changes to only HTML comments in the previous weeks, a single line of text was added to each page stating "This page was modified on *date and time*." This modest change seemed to encourage the WI to re-cache the altered pages because the following week 100% of the server files were once again recovered.

When the repository was taken off-line at the beginning of week 19, 100% of the server components were recoverable at the end of the week and continued to be recoverable through week 36 due, mainly because of Live's contribution. Google and Yahoo contributed far fewer resources after week 19 because most of the missing repository content was purged from their caches. Live slowly purged their caches, finally emptying them by week 43. IA contributed mainly CSS files each week (with a notable gap between weeks 30–33).

5. DISCUSSION AND FUTURE WORK

The injection technique using erasure codes seems to have been very effective for the proof-of-concept. Even though the Monarch Repository had never been crawled before, nearly all of its server components were recoverable within two weeks of it going live. This means the Monarch Repository would have lost *none* of its dynamically-produced web pages had it been lost whereas the repositories from Table 2 would have lost an average of 13% of their pages. Seventeen weeks after the repository was "lost," all the server components remained recoverable.

As mentioned earlier, injecting server components into the WI has some disadvantages. First, it puts an additional load, however small, onto the web repositories that they may not want to take-on. If the injection technique were adopted widely, web repositories may take steps to remove suspicious comments from crawled pages. Second, the additional payload to each page makes them download slower. This is certainly an issue where bandwidth is limited (e.g., developing countries and mobile clients). Third, some alteration of the visible contents of injected pages may be required to induce the WI to refresh its holdings. Forth, setting up such an injection mechanism requires a small amount of work by the webmaster before the website is lost; this is opposed to the lazy preservation approach of "no work required." And finally, it may not be wise to place private data into publicly accessible locations like the WI, even if encryption is used.

Despite these limitations, there is a need for a safety net like lazy preservation for preserving the server components of a lost website so re-enabling its functionality and accessing its deep web resources is possible. Perhaps a business model could be developed that provided an insurance policy for lost websites. An organization like Google with large amounts of disk space (and large amounts of public trust) could automatically back-up any number of web servers without cost to the webmasters. In the advent of a loss, the organization could recover the lost web server components for a fee. Considering the cost of re-building a dynamic website from scratch, a webmaster may be willing to pay a large amount of money to recover a lost website, enough to cover the collective cost of storing so much data. For such a mechanism to work, it would need to be easily enabled by the "laziest" of webmasters.

This was a proof-of-concept study– there remain many areas of investigation before this approach can be widely deployed. First, different methods of injecting the encoded pieces into the HTML pages need to be compared. We simply wrote a script that modified the static HTML pages, but various run-time inclusion methods (e.g., Apache Modules, server side includes, URL rewriting) should be evaluated. Second, the relationship between frequent updates to both the server-side components and the client-side resources needs to be further explored. In our experiment, changes to both server- and client-side resources were infrequent and highly synchronized. Third, we have implicitly assumed that the WI has not removed or corrupted the HTML comments. Our techniques would fail if the WI adopted an adversarial position. Finally, although the WI migrates client-side resources to new formats, it would not do so for server-side components. Something between the pessimistic and optimistic approaches needs to be developed that can express software dependencies and migration paths in a light-weight manner.

6. CONCLUSIONS

Several techniques for recovering the server components of a website from the WI were examined in this paper. One

promising technique was implemented in an EPrints repository, and it was demonstrated that nearly all the repository's server components could be recovered from the WI just two weeks after the repository was made accessible on the surface web. Pages were refreshed in the WI a few weeks after they were modified, allowing the modified server components to be completely recoverable. This injection technique may not be ideal for every type of website, but it does demonstrate that the lazy preservation approach to preserving server components is at least feasible with a small amount of work on behalf of the webmaster.

7. ACKNOWLEDGMENTS

This work is supported in part by NSF Grant IIS-0610841.

8. REFERENCES

[1] S. Abiteboul, G. Cobena, J. Masanes, and G. Sedrati. A first experience in archiving the French Web. In *Proceedings of the 6th European Conference on Research and Advanced Technology for Digital Libraries*, pages 1–15, Sept. 2002.

[2] A. Arvidson, K. Persson, and J. Mannerheim. The Kulturarw3 Project - The Royal Swedish Web Archiw3e - An example of "complete" collection of web pages. In *Proceedings of the 66th IFLA Council and General Conference*, Aug. 2000. `http://www.ifla.org/IV/ifla66/papers/154-157e.htm`.

[3] R. Baeza-Yates, C. Castillo, and E. N. Efthimiadis. Characterization of national web domains. *ACM Transactions on Internet Technology*, 7(2):9, 2007.

[4] M. K. Bergman. The deep web: Surfacing hidden value. *The Journal of Electronic Publishing*, Aug. 2001. `http://www.press.umich.edu/jep/07-01/bergman.html`.

[5] A. Cantrell. Data backup no big deal to many, until... *CNNMoney.com*, 2006. `http://money.cnn.com/2006/06/07/technology/data_loss/index.htm`.

[6] Consultative Committee for Space Data Systems. Reference model for an open archival information system (OAIS). Technical Report CCSDS 650.0-B-1, 2002.

[7] L. P. Cox, C. D. Murray, and B. D. Noble. Pastiche: Making backup cheap and easy. *SIGOPS Operating Systems Review*, 36(SI):285–298, 2002.

[8] Eprints for digital repositories. `http://www.eprints.org/`.

[9] C. Hank, S. Choemprayong, and L. Sheble. Blogger perceptions on digital preservation. In *Proceedings of JCDL '07*, page 477, 2007.

[10] E. D. Karnin, J. W. Greene, and M. E. Hellman. On secret sharing systems. *IEEE Transactions on Information Theory*, 29(1):35–41, 1983.

[11] S. Katzenbeisser and F. A. Petitcolas, editors. *Information Hiding Techniques for Steganography and Digital Watermarking*. Artech House, Inc., Norwood, MA, USA, 2000.

[12] M. Koster. A standard for robot exclusion, June 1994. `http://www.robotstxt.org/wc/norobots.html`.

[13] C. Marshall, F. McCown, and M. L. Nelson. Evaluating personal archiving strategies for Internet-based information. In *Proceedings of IS&T Archiving 2007*, pages 151–156, May 2007. arXiv:0704.3647v1.

[14] J. Masanès. Web archiving methods and approaches: A comparative study. *Library Trends*, 54(1):72–90, 2005.

[15] F. McCown. Windows Live Search development forum: Image search with 'site:' operator, June 2007. `http://forums.microsoft.com/MSDN/ShowPost.aspx?PostID=1799762&SiteID=1`.

[16] F. McCown, N. Diawara, and M. L. Nelson. Factors affecting website reconstruction from the web infrastructure. In *Proceedings of JCDL '07*, pages 39–48, June 2007.

[17] F. McCown, X. Liu, M. L. Nelson, and M. Zubair. Search engine coverage of the OAI-PMH corpus. *IEEE Internet Computing*, 10(2):66–73, Mar/Apr 2006.

[18] F. McCown, C. C. Marshall, and M. L. Nelson. Why websites are lost (and how they're sometimes found). *Communications of the ACM*, 2008. To appear.

[19] F. McCown and M. L. Nelson. Evaluation of crawling policies for a web-repository crawler. In *Proceedings of HYPERTEXT '06*, pages 145–156, 2006.

[20] F. McCown and M. L. Nelson. Characterization of search engine caches. In *Proceedings of IS&T Archiving 2007*, pages 48–52, May 2007. arXiv:cs/0703083v2.

[21] F. McCown and M. L. Nelson. Usage analysis of a public website reconstruction tool. In *Proceedings of JCDL '08*, June 2008.

[22] F. McCown, J. A. Smith, M. L. Nelson, and J. Bollen. Lazy preservation: Reconstructing websites by crawling the crawlers. In *Proceedings of WIDM '06*, pages 67–74, 2006.

[23] G. Mohr, M. Kimpton, M. Stack, and I. Ranitovic. An introduction to Heritrix, an archival quality web crawler. In *Proceedings of IWAW '04*, Sept. 2004.

[24] E. T. O'Neill, B. F. Lavoie, and R. Bennett. Trends in the evolution of the public web. *D-Lib Magazine*, 3(4), Apr. 2003.

[25] J. S. Plank. A tutorial on Reed-Solomon coding for fault-tolerance in RAID-like systems. *Software: Practice and Experience*, 27(9):995–1012, 1997.

[26] M. O. Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance. *Journal of the ACM*, 36(2):335–348, 1989.

[27] Registry of Open Access Repositories (ROAR). `http://roar.eprints.org/`.

[28] R. L. Rivest, A. Shamir, and D. A. Wagner. Time-lock puzzles and timed-release crypto. Technical Report TR-684, Cambridge, MA, USA, 1996.

[29] A. Traeger, N. Joukov, J. Sipek, and E. Zadok. Using free web storage for data backup. In *Proceedings of StorageSS '06*, pages 73–78, 2006.

[30] A. Waugh. The design of the VERS encapsulated object experience with an archival information package. *International Journal on Digital Libraries*, 6(2):184–191, Apr. 2006.

[31] B. Wu and B. Davison. Cloaking and redirection: A preliminary study. In *Proceedings of AIRWeb '05*, May 2005.