Just-In-Time Recovery of Missing Web Pages

Terry L. Harrison and Michael L. Nelson
Department of Computer Science
Old Dominion University
Norfolk, VA USA
{tharriso, mln}@cs.odu.edu

ABSTRACT

We present Opal, a light-weight framework for interactively locating missing web pages (http status code 404). Opal is an example of "in vivo" preservation: harnessing the collective behavior of web archives, commercial search engines, and research projects for the purpose of preservation. Opal servers learn from their experiences and are able to share their knowledge with other Opal servers by mutual harvesting using the Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH). Using cached copies that can be found on the web, Opal creates lexical signatures which are then used to search for similar versions of the web page. We present the architecture of the Opal framework, discuss a reference implementation of the framework, and present a quantitative analysis of the framework that indicates that Opal could be effectively deployed.

Categories and Subject Descriptors

H.3.7 [**Digital Libraries**]: System Issues

General Terms

Algorithms, Design, Experimentation, Human Factors

Keywords

Digital Preservation, 404 Web Pages, Apache Web Server

1. INTRODUCTION

Digital preservation projects typically involve controlled environments and collections and do not view the Web Infrastructure as a "living" preservation mechanism. We define Web Infrastructure (WI) to be the collection of commercial web search engines (Google, Yahoo, MSN, etc.), web archives operated by non-profit companies (e.g. the Internet Archive's "Wayback Machine") and research projects (e.g. CiteSeer and NSDL). We present a "just-in-time" approach to preservation which relies on the "living" web for "in vivo" preservation. It is not guaranteed by an in-depth institutional commitment to a particular archive, but achieved by the often involuntary, low-fidelity, distributed efforts of millions of individual users, web administrators and commercial services. Although the WI does not yet offer emulation, document refreshing and migration occur naturally, if somewhat randomly, on the "living web". Figure 1 shows the WI refreshing and migrating web documents, frequently as side-effects of de-duping efforts and user services. The original document was published in 1993 as a compressed PostScript (.ps.Z) file on a "nasa.gov" machine. The WI has since migrated it to other formats not available in 1993 (e.g., PDF & PNG) and refreshed it to machines all over the world.

We define a framework for harnessing the collective behavior of WI to locate web pages that are no longer present (http error 404). We are not concerned about the performance of any single member of the WI (they may come and go). The purpose of this research is to investigate the aggregate performance of the WI to see if it can be used to address the problem of missing web pages. An example scenario would be if a colleague recently left one university for another, you might find that your bookmarked URL results in a 404 error. A copy of the site may be in Google's cache, but this will not reveal their new telephone number, email address, etc. A Google search may turn up their new home page, but their name alone might not be sufficient to locate their new page if they have a very common name (e.g., "Michael Nelson"). Opal seeks to bridge this gap by finding the relocated page, using a lexical signature derived from the cached copy, and thus preserve a link from the old to new copy.

2. BACKGROUND

Despite well-known guidelines for creating durable URLs [3], missing pages (http error code 404) remain a pervasive part of the web experience. Kahle reported the expected lifetime of a web page is 44 days [11]. Koehler [12] performed a longitudinal study of web page availability and found the random test collection of URLs eventually reached a "steady state" after approximately 67% of the URLs were lost over a 4-year period. Lawrence et al. [15] report on the persistence of URLs that appear in technical papers indexed in CiteSeer. For recently authored papers, they found over 20% of the URLs listed were invalid. However, manual searches were able to reduce the number of 404 URLs to 3%. Spinellis [25] did a similar study for papers published in Communications of the ACM and IEEE Computer and found 27% of the URLs referenced therein were unavailable after 5 years.

HT'06, August 22–25, 2006, Odense, Denmark.

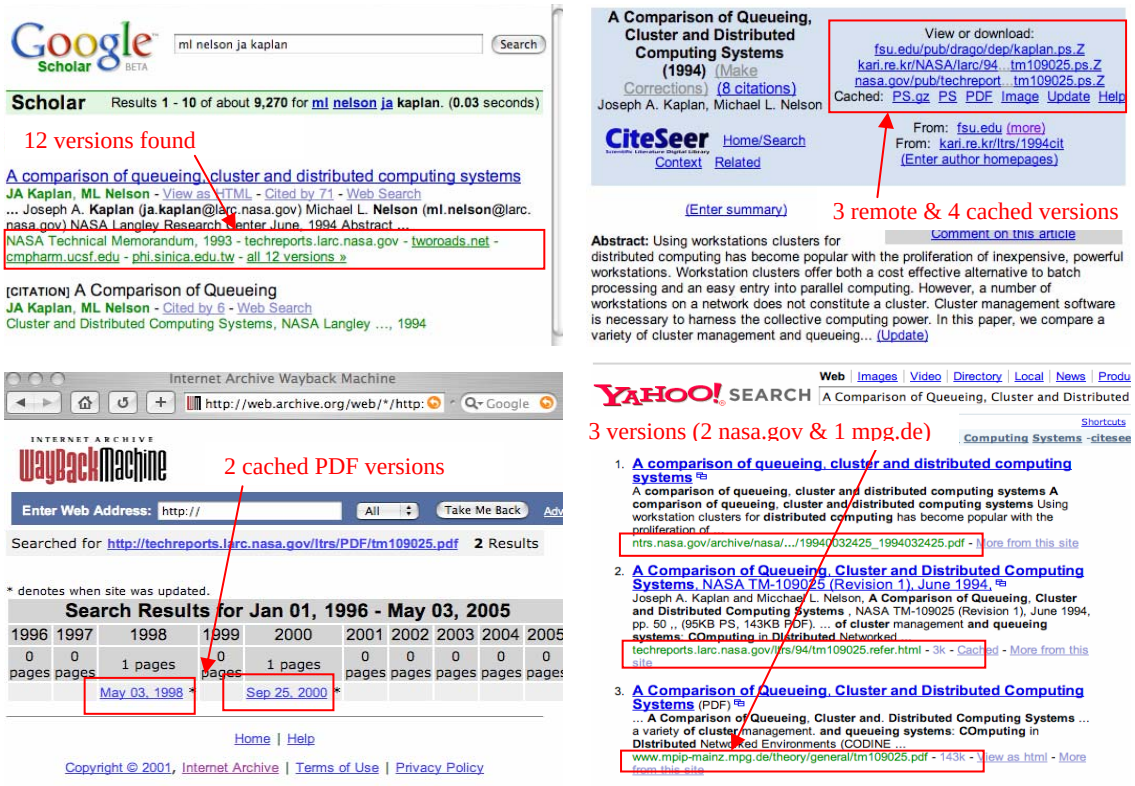

Figure 1: Refreshing and migrating occurring in the living Web

McCown et al. [17] focused on articles in D-Lib Magazine found a 10 year half-life. Nelson and Allen [19] studied object availability in digital libraries and while many URLs had changed during the period, manual searching found 3% of the URLs were unavailable after 1 year.

To address the problem of missing web pages, our solution relies on three key technologies: web engineering, lexical signatures, and the Open Archives Initiative Protocol for Metadata Harvesting. The mechanics of web servers are well understood and only discussed in passing in this paper; the latter two technologies are now introduced.

2.1 Lexical Signatures

A lexical signature (LS) is a small set of words that capture the "aboutness" of a document. It can be thought of as an extremely abbreviated metadata description of a document. Phelps and Wilensky [22] first proposed the use of LSs for finding content that had moved from one URL to another. Their claim was "robust hyperlinks cost just 5 words each" and their preliminary tests confirmed this. The LS length of 5 terms was chosen somewhat arbitrarily. Phelps and Wilensky proposed appending a LS as an argument to a URL:

```
www.cs.odu.edu/~tharriso/?lexical-signature=
terry+harrison+thesis+jcdl+awarded
```

where the LS is everything after the "?" in the URL. The idea is that most applications ignore such arguments if they are not expecting them (there are similar syntax tricks for appending a LS for an application that does expect arguments). They conjectured that if the above URL was 404, the browser would then look at the LS appended at the end of the URL and submit that to a search engine such as Google to find a similar or new copy. While there are many people named "Terry Harrison" in the world, there are far few who are doing digital library research, and this LS would help find either the new copy of the home page or related pages.

Although their early results were promising, there were two significant limitations that prevented LSs from being widely deployed. First, they proposed a scenario where web browsers would be modified to exploit LSs. Second, they required that LS be computed a priori. Park et al. [21] expanded on the work of Phelps and Wilensky, studying the performance of 9 different LS generation algorithms (and retaining the 5-term precedent). The performance of the algorithms depended on the intention of the search. Algorithms weighted for Term Frequency (TF; "how often does this word appear in this document?") were better at finding related pages, but the exact page would not always be in the top N results. Algorithms weighted for Inverse Document Frequency (IDF; "in how many documents does this word appear?") were better at finding the exact page but were susceptible to small changes in the document (e.g., when a misspelling is fixed). Table 1, shows two different LSs created from the author's home page. The first is TF weighted, and when submitted to Google, results in 11700 pages. The second LS incorporates IDF weighting in addition to TF, and when submitted to Google only results in three pages.

Lexical Signature	Calculation Technique	Results from Google
2004+terry+digital+harrison+2003	TF Based	11700
terry+harrison+thesis+jcdl+awarded	TFIDF Based	3

Table 1: Different Lexical Signatures for the same page.

Both result sets include the correct homepage, however the second LS is more precise. Park et al. approximated the IDF values when evaluating their test collection of web documents by mining Google. Though they ran their tests twice over a 10 month period, they did not explicitly investigate how LSs evolve over time.

2.2 OAI-PMH

The Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH) represents a six "verb" protocol for the harvesting of metadata records from digital libraries [13, 27]. The goal of OAI-PMH is to provide a low-barrier framework for to promote interoperability among digital libraries and is intentional simplistic in its design. OAI-PMH utilizes the concept of metadata harvesting to replace earlier "distributed searching" models that were popular among digital libraries. Responsibilities are separated into two distinct classes of participation, the "repository" and "harvester". Repositories provide metadata records which can be harvested. Harvesters gather records from repositories and provide value added services on their holdings. A harvester can act as a repository and can perform as an content aggregator in this way, if it has harvested from several repositories.

The six "verbs" that make up the protocol define the communications used to harvest from Repositories. Of these, "Identify, ListSets, and ListMetadataFormats" are used to provide information about the Repository's archive. "Identify" supplies data such as the archive's name and administrative contact information, and includes an optional "friends" container, to permit discovery of additional repositories. "ListSets" and "ListMetadataFormats" requests permit discovery of categories of records which are available. "ListIdentifiers", "GetRecord", and "ListRecords" are used to retrieve record identifications, single record, or multiple record retrieval, respectively. Additional arguments provide mechanisms to limit the number of records returned to certain metadata formats, and optionally to specified date ranges and "set" affiliations, as described in the OAI-PMH data model [14]. While originally designed to support scholarly eprint servers, there has been increased interest in applying OAI-PMH to a wider variety of usage scenarios, including thesauruses, usage logs, and as an OpenURL registry [28].

3. RELATED WORK

The LOCKSS ("Lots of Copies Keeps Stuff Safe") project is an elegant approach for distributing content between known participants [16]. LOCKSS is a collection of cooperative, deliberately slow-moving caches at participating libraries and publishers. The purpose of the caches is to provide an electronic "inter-library loan" if any participant loses files. The caches are slow-moving to both avoid overloading the network and to resist attacks. While LOCKSS is an attractive and inexpensive solution for libraries to replicate content from publishers' web sites, it is not immediately useful for in vivo preservation. This is because LOCKSS is designed to allow libraries to crawl the websites of publishers and ingest that content into their cooperating caches. There is an implicit model of how the data flows: from publishers to libraries. Given this limited scope, it is very unlikely that any library participating in LOCKSS would have the resources necessary to undertake general web crawling.

Previous work on finding similar web documents has used a variety of different techniques. Because of scalability concerns, traditional cosine similarity is often not used to generate all-to-all document comparisons because it is $O(n^2)$. A popular scalable technique is to use "fingerprinting", where documents are split into fingerprints, and if two documents share more than a threshold of fingerprints, they are considered related or the same. Fingerprinting has been used to detect plagiarism [4] and to identify duplicate and near-duplicate web pages for web crawler optimization [23, 5]. The most common replications they found were manuals for Java, Linux, web servers and the like. These papers are set in the context of improving the performance of a single WI member by some combination of skipping web pages that have already been crawled, increase the rank of pages that are duplicated (i.e., duplication is similar to linking in determining "importance"), and improving search result times. There is no notion of exploiting this knowledge outside the scope of a single search engine.

The Walden Path Manager (WPM) is a tool that evaluates changes in web pages and evaluates the suitability of replacement candidates [7, 6]. The goal is slightly different than Opal, and addresses the question "is this page still about elephants?" more than "is this a variant of the page authored by X on Y date?", but it still has the idea of creating a signature for each page and measuring the change for page Z to Z'. WPM must be installed on the local machine where the web page resides. It pre-analyzes the hyperlinks of interest from the page to be monitored, as soon as they are identified by the WPM administrator. Page signatures are generated using phrases rather than terms and IDF values are calculated via search engine queries. This gives the system some useful a priori knowledge when a 404 finally happens, but required the pre-selection of all the links of interest ahead of time. The system is not designed to share information with other instances of WPM.

In related research, we have studied the crawling and caching behavior of various members of the WI [24, 18]. We demonstrated that useful services can be built on top of WI caches, including the ability to reconstruct entire lost web sites. We also measured the time required for web resources to be discovered and cached (10-103 days) as well as how long they remained in cache after deletion (7-51 days). Opal depends on the long cache retention policies of various WI members to generate LSs after pages are discovered to be missing. Opal will work especially well for high-traffic pages that go 404, but risk increases as a page's popularity decreases.

4. OPAL

Although Phelps and Wilensky's original deployment plans (manual insertion, proxies that generate LSs, modified browsers, etc.) for LSs never came to pass, LSs can be deployed in a less intrusive manner. Consider a small number of distributed, cooperating Opal servers, O_1 .. O_n. We then take advantage of the fact that most web servers allow for custom pages (local and remote) to be returned for various status codes. This is accomplished on Apache servers by editing the configuration file ("httpd.conf") with the name of the document to handle the 404 error code:

```
ErrorDocument 500 "Abandon All Hope"
ErrorDocument 402 http://www.remote.com/subscribe/
ErrorDocument 404 /404.html
```

The goal is to make it as easy as possible for web server administrator to take advantage of Opal servers. This is done by adding a very simple JavaScript page tag to a web server's custom 404 page. The script is simple and easy to visually inspect:

```
<html>
<head>
<meta http-equiv="content-type" content="text/html">
<title>404 JavaScript Page</title>
</head>
<body>
<script language="javascript" type="text/javascript">
OpalServer="http://opal.foo.edu/opal.pl?new_url="
window.location=OpalServer+document.URL
</script>
</body>
</html>
```

By adding a JavaScript page tag to this custom 404 page, interactive users will then be redirected from a web server W to a Opal server, either to a specific O_i or a random O_i via a DNS rotor. *This page tag is the only configuration administrator for W will have to perform to use a remote Opal server.* Robots will not execute the page tag and will see just the 404 http error. This JavaScript page tag will pass the missing URL on W as an argument to the Opal server. Although the httpd.conf can be configured to redirect the user to a remote page, there is no way for the remote server to know the original URL that caused the 404. The JavaScript page tag is used to push the 404 URL as an argument to the remote Opal server.

Once a Opal server receives the 404 URL, it will check to see if has seen that URL before. If it has, it will present to the user a list of new URLs that have previously been found. It will also check the WI for cached copies of the URL from which to generate a LS. If we cannot find a cached copy in any portion of the WI, and no Opal server has mapped the old URL to a set of new URLs or has a previously calculated LS, then our interactive user is out of luck. However, if we have a cached version, we will offer the user the choice of viewing the cached copy, looking for a new version of the same page in the web (IDF-weighted LS). There are use cases for each scenario. If you wanted to contact a colleague that had recently left one university for another, their old page at the previous university would not be what you want. Similarly, if the colleague had retired, perhaps a similar page search would indicate who has assumed the retiree's responsibilities. All of these scenarios assume that the interactive user had some expectation of what the missing link to W would contain. Opal will not simply generate links for users to follow; it will also monitor their actions to mine the collective opinion of users to find the "right" new URL(s) for the 404 URL. Clicking on a page will send a message back to the Opal server with a one-time identifier to register the user's "vote" for the "right" page. The collective votes are tallied and used to rank options provided to future users.

4.1 Design Considerations

New Opal servers are introduced through a two-level web of trust model. First, the administrator of a new Opal server, O_{new}, must contact out-of-band (email, phone, etc.) the administrator of an existing Opal server O_{exist}. After the credentials of O_{new} are established, O_{exist} adds O_{new} to its "friends" container of the OAI-PMH Identify response. "Friends" is the OAI-PMH manner in which repositories can "point" to each other. Once O_{exist} adds O_{new} to its friends list, all the other Opal servers will learn of its addition on their next harvest. Figure 2 shows an OAI-PMH Identify response listing "friends" of Opal1.com.

```
<?xml version="1.0"?>
<OAI-PMH
      xmlns="http://www.openarchives.org/OAI/2.0/"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.openarchives.org/OAI/2.0/
        http://www.openarchives.org/OAI/2.0/OAI-PMH.xsd">
    <responseDate>2005-08-02T17:38:08Z</responseDate>
    <request verb="Identify">http://purl.lanl.gov/NET/srepod/www.cs.odu.edu/~tharriso/uHarvest.xml</request>
    <Identify>
      <repositoryName>Invivo2 URL Static Repository</repositoryName>
      <baseURL>http://purl.lanl.gov/NET/srepod/www.cs.odu.edu/~tharriso/uHarvest.xml</baseURL>
      <protocolVersion>2.0</protocolVersion>
      <adminEmail>tharriso@cs.odu.edu</adminEmail>
      <earliestDatestamp>2005-05-24</earliestDatestamp>
      <deletedRecord>no</deletedRecord>
      <granularity>YYYY-MM-DD</granularity>
      <description>
        <friends
          xmlns="http://www.openarchives.org/OAI/2.0/friends/"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://www.openarchives.org/OAI/2.0/friends/
            http://www.openarchives.org/OAI/2.0/friends.xsd">
          <baseURL>http://purl.lanl.gov/NET/srepod/www.cs.odu.edu/~tharriso/uHarvest.xml</baseURL>
          <baseURL>http://purl.lanl.gov/NET/srepod/www.cs.odu.edu/~tharriso/term_sr_Invivo2.xml</baseURL>
          <baseURL>http://opal2.com/terms.xml</baseURL>
          <baseURL>http://opal2.com/urls.xml</baseURL>
          <baseURL>http://opal3.com/terms.xml</baseURL>
          <baseURL>http://opal3.com/urls.xml</baseURL>
        </friends>
      </description>
      <description>
        <gateway
          xmlns="http://www.openarchives.org/OAI/2.0/gateway/"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://www.openarchives.org/OAI/2.0/gateway/
            http://www.openarchives.org/OAI/2.0/gateway.xsd">
          <source>http://www.cs.odu.edu/~tharriso/thesis/uHarvest.xml</source>
          <gatewayDescription>http://www.openarchives.org/OAI/2.0/guidelines-static-repository.htm</gatewayDescription>
          <gatewayAdmin>Patrick.Hochstenbach@ugent.be</gatewayAdmin>
          <gatewayURL>http://purl.lanl.gov/NET/srepod</gatewayURL>
        </gateway>
      </description>
    </Identify>
</OAI-PMH>
```

Figure 2: Identify Response for Opal1.com (With Friends).

Since the Web is ever evolving, the data Opal collects ages over time. All term IDFs and similar URLs (including metadata) are datestamped when they are learned by an Opal server. While the prototype does not implement this, one could envision a feature such that an Opal administrator could set age thresholds which remove older data from consideration in results processing. The harvesting process (an Opal server gathering data from other Opal servers) does provide the opportunity to update aging data points for term IDFs and similar URL data. As terms are harvested from another Opal server, they are compared against the current dictionary. If the term is new to the Opal server, it is simply added. If the term already exists in its database, the date of the harvested term and the one currently in the

dictionary are compared (at day granularity). If the new term has a newer datestamp, that means its IDF (and other associated data) has been calculated more recently. This harvested entry would then overwrite the current term entry. In such a fashion, harvest ingestion helps to maintain the freshness of the dictionary. Even though the document frequencies scraped from the WI for term IDF calculations are not likely to change quickly, keeping Opals IDF database fresh is a good policy. The process is identical for similar URL data.

4.2 Architecture

Figure 3 depicts the Opal server architecture. The steps are:

1. A user requests a web page (URL X) from a web server and it is not found (404);
2. A custom 404 page is sent back to the client with a JavaScript pagetag;
3. The pagetag redirects the user to the Opal server and passes the URL X as an argument;
4. Opal searches the WI for cached versions, generates a LS if one does not exist, and examines its own records to find similar URLs that have been voted on;
5. Finally, the options are returned to the user and they select how they would like to continue their exploration.

If the user selects the description of one of the resulting potential pages, this selection acts as a vote, indicating a human decision that this page is related. Opal records the vote, opens a new window and then issues a browser redirect to the desired page. The results of a subsequent vote tally will provide a ranked list of recommended alternate sites. These can be displayed to subsequent users who request the same 404 page. Figure 4 shows the interface presented to the user when Opal is engaged. It lists the missing URL, similar pages (as determined by the votes of others), any cached versions, and the ability to initiate a search to look for live versions (using a LS from a cached version). After Opal searches for the existence of current caches, successful results are displayed in the "Caches" section along with any LSs that Opal may have previously calculated for the 404 URL.

Caches provide the user with their first glimpse of content related to the 404 URL and such retrieval may satisfy the users retrieval needs. For example, if you were looking for a cake recipe, then finding a cached copy that may be entirely satisfactory. For cases where the current, non-cached site is required, this provides the user the opportunity to select the most appropriate cached content with which to continue their search. The LS is either manually selected by the user if a matching LS is already known by Opal, or dynamically calculated from the copies found in the WI caches. The user can then choose the search engine (Google, Yahoo, MSN, etc.) in which to submit the LS.

4.3 Opal Framework

The Opal framework consists of one or more collaborating Opal servers. Using Opal requires three components: the 404 redirect entry in the httpd.conf file of the participating server, the custom 404 HTML page with the JavaScript tag on the participating server, and the actual Opal script running on a remote server. The Opal server contains implementation specific storage structures hold learned data records, including term IDF values and 404 URL voting histories. The details are discussed in [9], but at a high level it maintains a "Term Db" that maps Terms → IDF values, and a "URL Db" that maps 404 URLs → similar URLs and similar URLs → {datestamps, votes, metadata}. A set of support scripts permit an Opal server to act as an OAI-PMH repository. By exposing records and harvesting records from other Opal servers, each server increases its local knowledge. This reduces the number of external queries required to calculate results for client requests and promotes more efficient response times.

Once the an initial request is made, a variety of lookups are conducted to pull together the data needed for the response. A Lexical Signature lookup in the URL Db the will determine if any previously calculated LS exists that corresponds to the 404 URL. A URL lookup will provide those URLs which have acquired votes in association with the URL in question. The top N similar URL results are then queried in the URL Db for their descriptive metadata. In the current implementation, external searches are also conducted at this stage, against search engines (Google and Yahoo) to determine if a hyperlink to a cached version can be located. This is accomplished by simply concatenating the 404 URL to each respective engine's search query URL string, as shown in the following Google and Yahoo queries:

```
http://www.google.com/search?hl=en&ie=ISO-8859-1&
q=http://www.cs.odu.edu/~tharriso
```

```
http://search.yahoo.com/search?fr=FP-pull-web-t&
ei=UTF8&p=http://www.cs.odu.edu/~tharriso
```

If the Opal server locates an LS and the user opts to search for new paths using it, then it is fed directly to the selected search engine for the result set. The following URL shows a Google search using the LS.

```
http://www.google.com/search?hl=en&ie=ISO-8859-1&
q=terry+harrison+thesis+jcdl+awarded
```

If the user chooses to explore using a cached version, the cache content is retrieved and parsed. Opal calculates TF and queries its Term Db for each terms IDF weight. If a term's IDF is not available here, then a search engine is queried to ascertain its value. This IDF data is appended to the Term Db to reduce the overhead of subsequent lookups. Favoring precision over recall, this implementation does not stem the terms.

We currently mine Google for IDF values for the web. Whereas in the future we might use the Google API, we are currently just screen scraping the IDF values from the Google search page (for example, see Figure 5). We have assumed the size of the corpus as eight billion web pages, as reported on Google's splash page. While these values are just estimates, they have provided good results in our preliminary usage.

Once all TF and IDF values are retrieved, then the standard TFIDF calculation is performed for each term from the cache and the LS determined (from the top 5 results).

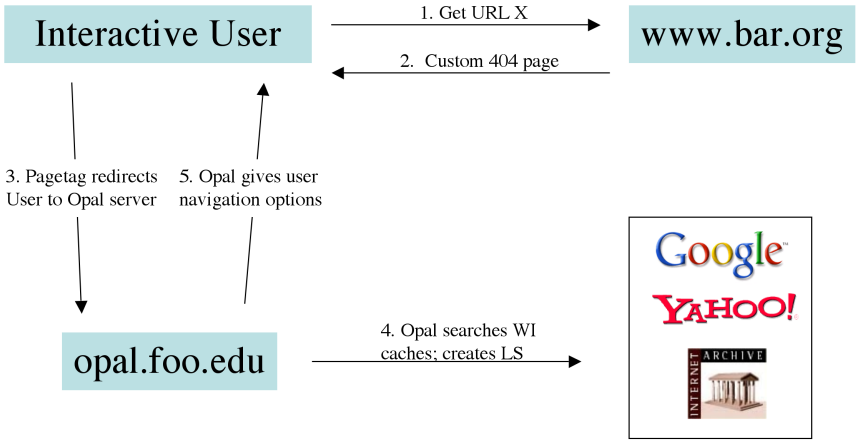

Figure 3: Opal Architecture

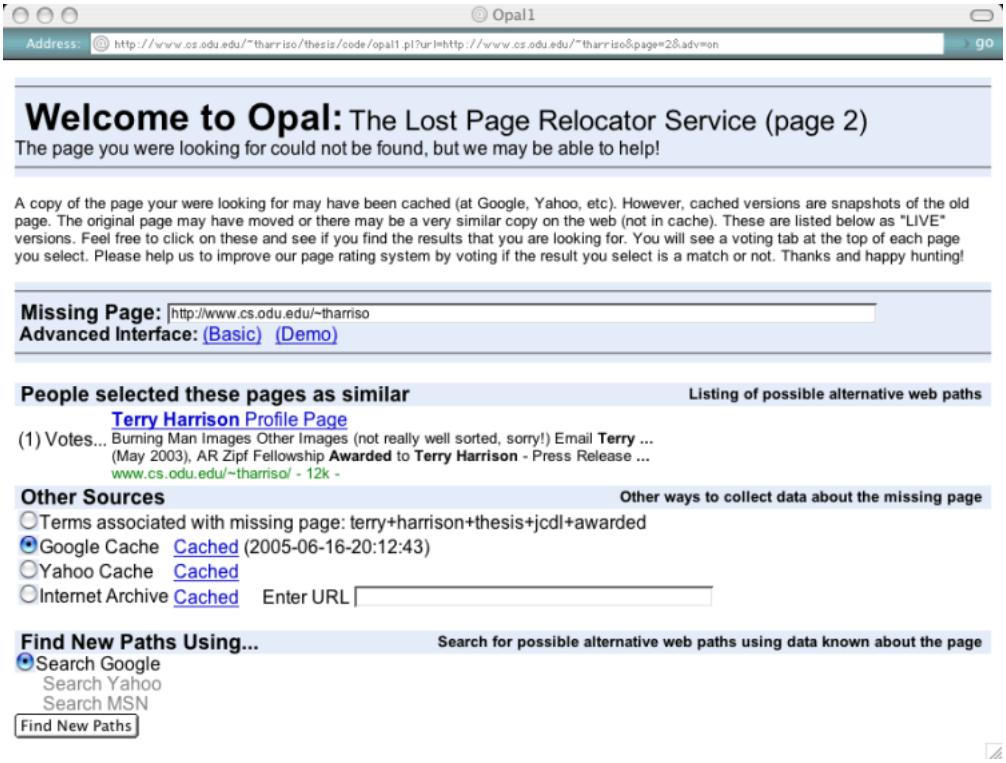

Figure 4: Opal Interface for Page www.cs.odu.edu/~tharriso/

Figure 5: Screen Scraping IDF Values From Google

The LS is stored in the URL Db and is also used to drive the selected search engine for new paths (as we saw earlier when a pre-existing LS was selected). The resulting URLs are recorded in the URL Db along with their descriptions, as presented by the search engine.

The new URLs are presented to the user, but not associated in the URL Db with the 404 URL until after a user has voted on it. If an entry in the URL Db does not exist for the new found URL, then the URL and Descriptive metadata are simply added to the Db. This is considered a "New Vote". If any does exist, the vote is considered a "Supporting Vote" and the tally for this 404 URL/New URL association is incremented. At the time of the vote tally, acquisition datestamps for descriptive metadata pertaining to a URL are compared with the freshest datestamp entry writing over any older entry. The user is presented with any URLs that Opal has already associated with the 404 URL and those that are discovered via interactive exploration with Opal. Descriptive metadata parsed from search engines or recovered from the URL Db provide the user with the same information as would result from search engine search. Armed with this, the user decides which links to explore and clicks on the hyperlink of the URL chosen to go to the desired page. Voting harnesses this cognitive association.

4.3.1 Client-Server Communication

Communications between the client and an Opal server occur using CGI over HTTP. Arguments are passed as queries, with query names and values separated by an equals sign (=). Queries are separated by an ampersand (&). Requests are passed using the http GET method, which makes them easier to bookmark for later retrieval. Several requests to an Opal server, (opal1) are provided and explained below. URL encoding has been removed to improve readability.

The following request shows only "new_url", and no "url" parameter, which means this is a request created by means of a server re-direct. Page 2 is the entry page for server re-directs and the GUI displayed should be Basic, since "adv" is set to NULL:

```
http://www.cs.odu.edu/~tharriso/thesis/code/opal1.pl
?adv=&new_url=http://www.cs.odu.edu/~tharriso&page=2
```

This complicated URL requests the Opal server to find new paths (page 3) for an unmodified URL (url == new_url) using the Internet Archive cached version residing at "ia_url" and then submit the resulting LS to Google:

```
http://www.cs.odu.edu/~tharriso/thesis/code/opal1.pl
?adv=on&demo=on&url=http://www.cs.odu.edu
&new_url=http://www.cs.odu.edu
&version=Cache&verbose=On&c=internetArchive&
ia_url=http://web.archive.org/web/20041118084857/
http://www.cs.odu.edu/&search=google&page=3
```

Clicking on one of the resulting links will open a new window and send 1 vote back to the Opal server for the relationship between "url" and "match":

```
http://www.cs.odu.edu/~tharriso/thesis/code/opal1.pl
?method=vote
&url=http://whiskey.cs.odu.edu/new_position.html
&match=http://www.cs.odu.edu/new_position.html
```

4.3.2 Server-Server Communication

If each Opal server were left to its own devices, it would slowly learn term IDF values and similar URLs for 404 URLs over time, yet this would require a great deal of work for the server and many search requests. Opal is scalable, both in finding and mapping old URLs to new URLs and generating LSs and IDF values from Google, because it is a collection of cooperating servers. We map each term and URL to an XML record. Since OAI-PMH is suitable for the transfer of any XML encoded data, we then use OAI-PMH interfaces for each of the Opal servers so they can harvest each other. O_i can harvest from O_j (and vice versa), to learn new IDF values and new URL mappings. If a network of Opal servers grew large enough, OAI-PMH aggregators could be used to create multiple hierarchies of servers, thus ensuring scalability. In Figure 6, Group 1 consists of four cooperating Opal servers, each acting as a OAI-PMH compliant repository. Each server (A, B, C, and D) harvests records directly from the other three. Graphically represented, this is a fully connected graph. Opal D is also a member of Group 2, (D, D.1, D.2, D.3). In addition to Group1 harvests, Opal D can also harvest from Group 2. Likewise Opal D acts as an aggregator providing other Group1 members with a access to Group 2 records and vice-versa. This enables a new Opal server to quickly bootstrap itself with harvested records, and keeps the server from having to issue requests to search engines that another Opal server have already performed. Harvesting can be performed by any OAI-PMH compatible harvester. Local harvest ingestion scripts would be tailored to the specific implementation details of the Opal server (Db, file system based, etc).

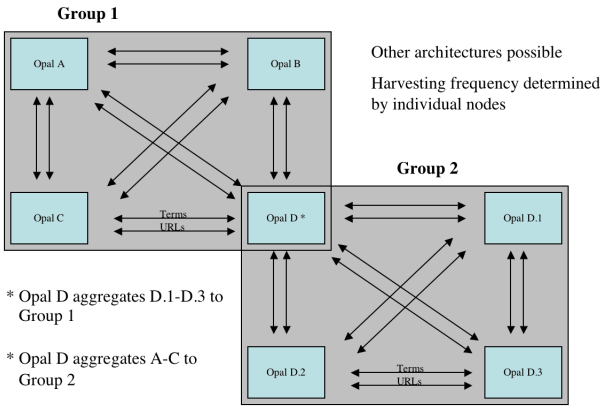

Figure 6: Opal Harvesting Hierarchy

When a potential matching URL is clicked, it is really a call to Opal to cast a vote for this 404 URL / New URL association and a request to display the page. For this prototype, the vote is appended to a log file and the file is

later parsed (tallied) via a cron job. Once the tally is concluded and the results combined with previous results, the URLs which were voted on can be displayed in the "Similar URLs" result set, without the need for the additional, expensive cache explorations. This data can also be shared with other Opal servers, to eliminate the need for redundant cache explorations.

XML Schemas are used by Opal to define unique OAI-PMH "metadataFormats" for export and harvesting of term and URL related datasets. Schemas were initially generated using XSDinference and then hand refined. The Term Schema defines the structure of XML formatted term data sets which are to be harvested. The URL Schema defines the structure of XML formatted URL data sets which are to be harvested. The schemas and examples are detailed in [9]. The Opal prototype exposes these XML structures through Static Repositories [10], which are registered at the Los Alamos National Laboratory Experimental OAI Static Repository Gateway (http://libtest.lanl.gov/). Static repositories allow for rapid prototyping, but their file size limitations of 2MB would prevent them from being used in large scale Opal implementations.

5. DISCUSSION

In the course of this work, many sites were tested. During this time it has been interesting to discover what terms are found in an LS. Table 2 shows LSs created by Opal from live sites. The first few were what we expected. We find the presence of unique items such as self referring URLs, email addresses, telephone numbers, and zip-codes. "the" was found in the 5th LS (digitalpreservation.gov) and is an artifact of the current IDF calculation mechanism. Google has many servers and their indexes are not all synchronized, making it difficult to determine the correct corpus size to apply in the calculation of IDF. For the prototype, the size which Google publicizes on its home page as the number of documents indexed, 8 billion, is used. This number is the total number indexed, and is greater than the number indexed on any single Google server. This discrepancy inflates the value of "stop words" just enough that they may make it into an LS in cases of large pages where their TF is high, since the LS is comprised of the top 5 ranked TFIDF terms for the document. While this could be refined, it is interesting to note that the LS still retrieved the desired site as the top ranking match. Sites like hot-deals.org have been the most interesting, where the LS seems quite generic, yet it is highly effective at locating the desired page.

Opal builds LSs based on term analysis, which rules out some pages as candidates for Opal. Flash sites and sites that make heavy use of images instead of text do not provide the term content needed for LS creation. Without an effective LS, Opal cannot locate matching pages. The use of HTML frames currently provides a similar challenge, though more advanced implementations could address this.

6. ANALYTICAL EVALUATION

Opal is a learning system, and over time it will attain more knowledge about term IDF values, cached locations, and similar pages for the 404 URLs it handles. Opal must go through a learning curve or bootstrapping process before it can gain enough data to reach a respectable quality of service which we will call the "steady state". Each time Opal makes an external query in order to fulfill a client request, a network overhead operational cost is incurred. Queries are used to dynamically locate links to currently held caches, learn term IDF values, and to search for similar URLs using an LS. $Cost_{Cache}$ is the number of requests required to located cached pages that are associated with the 404 URL.

$$Cost_{Cache} = (|S| * N) + R \quad (1)$$

S is the set of all services (e.g. Google, Yahoo, IA), N is the cost to locate a cache, and R is the cost to retrieve any cached copy datestamps. In the prototype, three services (Google, Yahoo, and IA) are checked at a cost of one network communication for each, with an additional communication to request the Google cache datestamp. Yahoo does not provide datestamp information and IA may contain multiple dates to select from. In the case of IA, Opal does not predetermine which cache to select. For the prototype, the $Cost_{Cache} = (|3|*1)+1 = 4$ network requests.

Finding new paths (or URLs) for the 404 URL requires retrieval cost of a pre-selected cache's content (R_c), individual IDF query cost for each unknown term (T) scraped from the cache, and a subsequent query cost (R_l) for using the LS, resulting in the $Cost_{Paths}$ formula:

$$Cost_{Paths} = R_c + |T| + R_l \quad (2)$$

For the prototype, the $Cost_{Paths} = 1 + |T| + 1$ or $|T| + 2$ network requests. The worst case network cost to serve a client request via cache location, term data gathering, and subsequent LS searching is the total of $Cost_{Cache}$ and $Cost_{Paths}$, which in our case, is $(4 + |T| + 2)$ or $(|T| + 6)$ requests. Let us say that there are 100 unique words on a given web page. If our Opal server had no prior knowledge, it would require 106 network requests to gather the data required to generate a response to the client. Opal learns and now has awareness of 100 term IDF values, which will not need to be queried again. Cache and LS search costs do not always exist, nor do they have to be paid by Google. Opal could store the cache location data for a period of time to remove the penalty for recalculation for subsequent requests for the same 404 URL. Fulfilling a client request will often include searching for new paths once the LS has been calculated. Any search engine could be used to retrieve results from a LS search multiple search engines could fulfill this request, reducing the penalty to Google, which is currently used to scrape IDF data. This leaves us to consider just the volume of term lookups.

6.1 Bootstrapping

The question arises, what is required during the bootstrap process for Opal to reach a steady state, in which we can expect a reasonable quality of service? We assume an Opal server can only make 1000 queries to Google per day, (we will assume this rule with other search engines as well), since this is a restriction placed on use of the Google API and whose spirit we will honor, even though we do not make use of the API. A steady state is achieved by an Opal server when it contains enough learned knowledge that it can fulfill all typical client queries on a given day. The number of unique term on the web is difficult to estimate. In the TREC Web Corpus WT10g collection there are close to 1.7 million documents consisting of 1.6 million unique words [2]. The Oxford English Dictionary contains about 250000 words or

URL	Lexical Signature
http://www.cs.odu.edu/∼mln	lloyd+nelson+http://www.cs.odu.edu/∼mln+dl+mln@cs.odu.edu
http://www.cs.odu.edu/∼mukka	tues+mukka@cs.odu.edu+683-3901+1300-1445
http://www.digitalpreservation.gov/	ndiipp+preservation+digital+the+congress
http://www.cs.odu.edu/∼cmo	simulation+cs+summer+powerpoint+spring
http://www.hot-deals.org	deal+deals+involving+see+coupon

Table 2: Lexical signatures created by Opal.

about 750000 if we consider distinct senses, and does not cover technical vocabularies or jargon. With a knowledge of one million terms, it may seem possible that an Opal server would not have to issue IDF data requests for new terms with great frequency. One interesting view is that ten Opal servers could divide up a list of these terms and could learn (and share) the IDF weights for these terms in 100 days. Increasing the number of servers decreases the time required. In learning via cache content scraping, Opal learns the words that occur the most frequently first. Since word frequency follows a "Zipf distribution" [1], we hypothesize that the rate at which Opal learns new terms will diminish quickly over time. Simulations were conducted to test scenarios that would help estimate Opal server learning rates. We want to see the differences in single server Opal learning rates, based on an assumed average number of terms per document with a Zipf distribution and with an even distribution (for comparison). A lexicon of 1 million terms was chosen for the simulation. In the "even" distribution, each term in the corpus appears with equal probability. In the "Zipf" distribution, the frequency of a term is proportional to its rank:

$$freq(r) = r^{-n} \tag{3}$$

In Zipf distributions n is near 1; for simplicity we chose $n = 1.0$. This means a few terms (e.g. "a", "an", and "the") will have a high probability of being selected, while more rarefied terms will have a very low probability of selection.

We ran tests with the following parameters and averaged the results of 100 iterations of each test:

- 1 Million terms assumed as number of terms in lexicon
- 30000 Document collection
- Even and Zipf distributions with 100 and 200 term documents

Figure 7 shows the number of new terms that Opal can learn from each document over the course of analyzing the document collection. As Opal increases its term awareness, there are fewer and fewer terms that it does not know. In the even distribution cases, Opal quickly learns the terms. Increasing the document size from 100 to 200 terms increased the rate of learning dramatically in this distribution. Simulations using the Zipf distribution (more accurately modeling the web) resulted in Opals rate of learning quickly dropping off per document. As an Opal server examines more documents, there are fewer new terms in them. Words like "a" and "the" are predominant and displace the rarer words which appear less frequently. Changing document size did not have as great a consequence with the Zipf distribution.

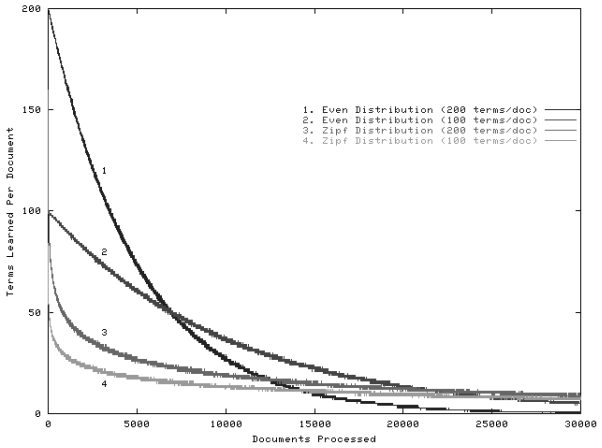

Figure 7: Terms Learned per Document

Figure 8 shows Opal's cumulative term learning. Even distributions clearly learn the most unique terms in the shortest time. Again with the Zipf case, document size did not have a great impact. This makes sense intuitively: large documents probably repeat many of the same words and have more stop words.

The even distributions are useful in providing an upper bound for Opal learning rates. Certainly, one could argue the pathologic upper bound case of a small set of documents (say 10000 with 100 words each) where each term found across the collection is unique, but chances of Opal running across such a case is highly unlikely. With the Zipf cases, the distribution is closer to what is found on the web and so we get a closer approximation of Opal performance. With Zipf cases, even after 30000 documents have been parsed, new terms are learned (Figure 8) at the rate of about 10 per document, with small changes anticipated in this rate, even with the ingestion of greater numbers of documents.

6.2 Scalability

Another way to approach the steady state issue is to consider the work load presented to the Opal server. To get an better grasp on the occurrence of 404 errors, we examined server logs from the Computer Science Department at Old Dominion University for the month of May 2005. Of 4.5 million http requests, there were approximately 500000 unique URL requests, and 75000 total 404 unique URLs. We learn that roughly 15% of the unique URL requests are for URLs which have gone 404. Rather than assume an even distribution of the total 404 errors across the number of unique 404 URLs, we looked further into the logs to get an idea of the distribution (details in [9]). The logs showed a power

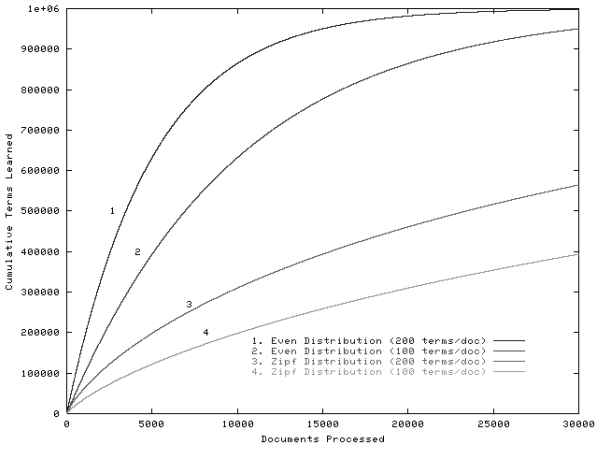

Figure 8: Cumulative Terms Learned

457817	Total 404 errors
74677	Total Unique 404 URLS (15%)
56157	75% Accesses From Interactive Users
5615	10% Cached or Already Have an LS
187	Daily Requests That Require Term Lookup

Table 3: Monthly Load for a Single Opal Server.

law distribution of the requests for the 404 URLs, where 25% of the totals were for the top entry, the "favicon.ico" (the small icon that appears in the browser URL bar, like the little Y! that appears when you go to yahoo.com). Fifth on the list is "robots.txt" which is not a "real" file either. The interesting thing to note from this is that such occurrences, represented over 25% of the logged 404 errors would not trigger Opal in most cases, since these requests are predominately the domain of crawlers, which will not follow the JavaScript re-direct which brings the user to Opal. Of the remaining requests, we must estimate the number of these URLs which Opal can work with. Not all 404 pages will have a cached copy available, which Opal will require at some point (either in a previous request, or in the current one) to calculate the LS. Let us assume that 10 percent of 404 URLs were at some point cached and are suitable Opal candidates. Putting together what we have gathered so far, we end up with about 200 request per day for Opal (Table 3). To handle about 200 requests per day means that the Opal server (limited to 1000 lookups to any given search engine) would have to be able to honor each request with an average of five lookups per request. If Opal had sufficient internal knowledge to meet these demands, then we could say that it is in a steady-state for the anticipated ODU workload. While in the bootstrapping itself, a single Opal server may not be able to handle the full load of 404 pages. Clients could be presented with a 404 error page and the 404 URL could be queued for background processing during periods of lighter activity. This type of inconsistent behavior may not be desired and suggests one of the other methods of bootstrapping for term learning, which does not involve direct human interaction.

With 10 Opal servers, each with the capacity to request 1000 IDFs per day, and each harvesting the other's data sets could learn a vocabulary of 1 million terms in about 3 years. Bootstrapping could also occur by searching for IDFs from a controlled dictionary, or via a harvest of another Opal server that has already ingested such a number of terms. An Opal server could bootstrap itself, even though this is not the most efficient way to start the learning process, or an Opal system could learn cooperatively as a team, which will expedite learning. Data sets could be harvested from an Opal server that has been running for a longer period of time, to jumpstart the new system. Controlled vocabulary lists could help with the initial learning process as well. Over time, and especially by sharing knowledge, it is likely that Opal servers can reach a state where term requests required on each Opal server fall within the permitted daily thresholds and the server can attain a "steady state". Simulations show that the number of anticipated lookups is not far off from numbers that are likely to be required for Opal servers to reach their "steady state".

6.3 Security Considerations

Security is always a concern for systems offering information services on the Web. Intentional attacks, such as intruders posing as Opal or man in the middle attacks, could be resisted though the use of encryption and session keys. Inadvertent attacks could also arise, in forms such as casting multiple votes during a session for a given page, or a denial of service due to a server reconfiguration that renders a large number of pages suddenly 404 and requiring Opal services. To avoid vote "double counting" votes could be registered with the voters IP in addition to the datestamp. Excessive votes on a 404 URL from an IP could be disregarded during tallying. To handle irregular spikes in traffic to Opal, cooperating Opal servers could employ load-balancing techniques. While it is not possible to say that all attacks, intentional or unintentional can be thwarted, measures could and would need to be employed to protect the integrity of Opal's services in a production environment.

7. FUTURE WORK

The Opal prototype demonstrated the potential of the Opal framework, but further work remains. Larger scale simulations could help test methods to reduce communication costs. User studies on alternate interfaces would increase user-friendliness of the system. Since the WWW is constantly evolving web sites that were located as similar in the past may have since moved themselves. It is possible, that pages may be restored to new locations after voting sessions have occurred from a pool of lesser candidates. To maintain accuracy, votes could be aged, and could be removed from the vote tallying process beyond a given threshold. Late arriving URLs (which have not had the chance to amass votes) could also be helped by random permutation of the result rankings [20].

7.1 Optimization

Prototyping of Opal revealed several areas that could be improved with a variety of code optimizations. In the Opal prototype, hashes tied to the file system and XML Static Repositories were used. While these work for prototyping purposes, they would need a more robust implementation for production level versions. Current Static Repositories at LANL are limited to 2MB in size and are not large enough to

contain large data sets for OAI based harvesting. Databases can be used to provide dynamic lookups instead of the use of tied hashes. OAI-PMH "Sets" could be used to distinguish data sets originating from different Opal servers. One limitation with Static Repositories is that they do not support the OAI-PMH notion of "Sets", which could be used to identify each Opal server's data sets and thus provide a means of minimizing redundant data harvesting [8]. If O_1 harvests $O_{A..C}$ while O_2 harvests $O_{C..G}$, then it might make more sense for O_3 to harvest O_1 and O_2 (even with the redundancy of the O_C data) rather than harvest O_1, O_D, O_E, O_F, and O_G separately. By using "Sets" O_3 could harvest from O_1 and then O_2, but request only sets D,E,F,G from O_2, so reducing the harvest redundancy and the subsequent de-duplication overhead.

A warning could be issued when a web page is not a good candidate for extracting an LS. Flash sites and others that are heavily image based, or otherwise lack significant textual content, are not good candidates for Opal. A mechanism could be developed that identifies and flags these types of sites and does not direct them to Opal. A standard or custom 404 error could be displayed in lieu of Opal. It would be helpful to provide a mechanism to handle situations where learned "similarURLs" for a 404 URL go 404 themselves. They could be temporarily removed from result set and quarantined. If the site were down beyond a specified period of time, its stored entry could be removed. Another option might be to display it in the results set with a 404 warning and let the user decide if they wish to pursue using Opal's services to relocate that page variant as well.

7.2 Lexical Signatures

The concept of lexical signatures is crucial to the operation of Opal but has not been studied extensively. Serious questions remain about the suitability of LSs for different file types. Park et al. [21] applied some heuristics (e.g., discarding documents with less than 50 terms), but did not study in depth the upper and lower bounds of where LSs are appropriate. A lower bound for term size is easy to conceptualize, but is there an upper limit? Is there an optimum document term size for using LSs? Should the length of the LSs grow as a function of a document's number of terms? How would LSs fare against "spammed" or "cloaked" pages (i.e., pages that do not " tell the truth" or modify their contents based on the requester)? LSs have not been explicitly investigated to see how they evolve over time. Intuitively, LSs for cnn.com or slashdot.org would be more difficult (if not impossible) to generate given the constantly changing contents of those pages. But what of the more subtle cases: when a researcher moves from one institution to another, how does the LS change? The person is the same, but the zip codes, email addresses and other uniquely identifying terms are likely to change.

7.3 IDF

We also plan to investigate additional IDF evaluation techniques, such as estimation based on limited crawls of hyperlinked neighboring pages [26]. Scraping Google for IDF values is not a viable long-term strategy, and at the very least we have not considered multi-lingual support in our prototype. Clearly any dictionary-based IDF technique will have to consider and weigh source languages. How will languages "cast" into Latin alphabets work? For example, is the name of Libya's leader "Khadafi" or "Qadafi"? It is essential to reduce the number of network communications required by Opal (on average) to fulfill client requests. Bootstrapping can be a demanding process, and provides fruitful grounds for exploration. The TREC-10 Web Corpus could provide a closer match to the IDF values that are based off of the actual web. Comparative research could determine how closely term frequencies from this corpus match with those reported by Google. If TREC could provide suitable IDF values, then the calculation of these could provide almost 2 million IDF values with which to bootstrap Opal.

7.4 Learning New Code

Finally, the APIs for many of the WIs are informal or obfuscated by design. Google has a formal API, but it requires registration and is limited to 1000 connections / day. It is likely that the URL structures employed by search engines, which Opal utilizes to access their respective caches will change frequently. Rather than installing new code at each Opal server, we plan to encode the APIs for each WI member as a web services definition language (WSDL) document. Since WSDL is encoded in XML, we can use OAI-PMH to harvest WSDL records from each server. We can attach XML signatures to the WSDL files to mark their authenticity. This would require only 1 trusted party to update a WSDL record on a single Opal server when an API changes and the update will propagate through the Opal server network when they harvest each other.

8. CONCLUSIONS

Opal is a flexible, light-weight framework for locating alternative pages that are similar to URLs that have gone 404. A few Opal servers are sufficient to service many web servers; configuring a web server to use an Opal server is as easy as installing a JavaScript page tag in a custom 404 error page. Opal takes advantage of the Web Infrastructure as a mechanism to locate caches, learn term IDF weights for the web, and to search for new pages using lexical signatures. Opal provides a service that encompasses many individual search services and leverages their collective knowledge, using the OAI-PMH to transport collective knowledge about similar URLs and lexical signatures. The Opal prototype demonstrates an effective system for relocating 404 data on the web. Even without optimizations, analytical evaluation shows that the network communication costs are not far off from those required for larger scale implementation and suggest the further research is warranted. Opal brings together services from the WI (search engines), lexical signatures, along with OAI-PMH as a communications vehicle to promote in vivo preservation. While not a substitute for a thoughtful preservation plan, Opal's ease of deployment should increase its chances of adoption. By locating candidate replacements for 404 web pages, we maintain the link between old and new copies in the living web.

9. REFERENCES

[1] L. A. Adamic and B. A. Huberman. Zipf's law and the Internet. *Glottometrics*, 3:143–150, 2002.

[2] G. Amati, C. Carpineto, and G. Romano. FUB at TREC-10 web track: a probabilistic framework for topic relevance term weighting. In *Proceedings of TREC-10*, pages 182–191, 2001.

[3] T. Berners-Lee. Cool URIs don't change. 1998. http://www.w3.org/Provider/Style/URI.html.

[4] S. Brin, J. Davis, and H. Garcia-Molina. Copy detection mechanisms for digital documents. *SIGMOD Record*, 24(2):398–409, 1995.

[5] J. Cho, N. Shivakumar, and H. Garcia-Molina. Finding replicated web collections. *SIGMOD Record*, 29(2):355–366, 2000.

[6] Z. Dalal, S. Dash, P. Dave, L. Francisco-Revilla, R. Furuta, U. Karadkar, and F. Shipman. Managing distributed collections: evaluating web page changes, movement, and replacement. In *JCDL '04: Proceedings of the 4th ACM/IEEE-CS joint conference on Digital libraries*, pages 160–168, 2004.

[7] L. Francisco-Revilla, F. Shipman, R. Furuta, U. Karadkar, and A. Arora. Managing change on the web. In *JCDL '01: Proceedings of the 1st ACM/IEEE-CS joint conference on Digital libraries*, pages 67–76, 2001.

[8] T. G. Habing, T. W. Cole, and W. H. Mischo. Developing a technical registry of OAI data providers. In *ECDL '04: Proceedings of the 8th European Conference on Research and Advanced Technology for Digital Libraries*, pages 400–410, 2004.

[9] T. L. Harrison. Opal: In vivo based preservation framework for locating lost web pages. Master's thesis, Old Dominion University, 2005.

[10] P. Hochstenbach, H. Jerez, and H. Van de Sompel. The OAI-PMH static repository and static repository gateway. In *JCDL '03: Proceedings of the 3rd ACM/IEEE-CS Joint Conference on Digital Libraries*, pages 210–217, 2003.

[11] B. Kahle. Preserving the Internet. *Scientific American*, 276(3):82–83, March 1997.

[12] W. Koehler. Web page change and persistence — a four-year longitudinal study. *Journal of the American Society for Information Science and Technology*, 53(2):162–171, 2002.

[13] C. Lagoze and H. Van de Sompel. The Open Archives Initiative: building a low-barrier interoperability framework. In *JCDL '01: Proceedings of the 1st ACM/IEEE-CS Joint Conference on Digital Libraries*, pages 54–62, 2001.

[14] C. Lagoze, H. Van de Sompel, M. L. Nelson, and S. Warner. The Open Archives Initiative Protocol for Metadata Harvesting. http://www.openarchives.org/OAI/openarchivesprotocol.html, 2002.

[15] S. Lawrence, D. M. Pennock, G. W. Flake, R. Krovetz, F. M. Coetzee, E. Glover, F. Nielsen, A. Kruger, and C. L. Giles. Persistence of web references in scientific research. *Computer*, 34(2):26–31, 2001.

[16] P. Maniatis, M. Roussopoulos, T. J. Giuli, D. S. H. Rosenthal, and M. Baker. The LOCKSS peer-to-peer digital preservation system. *ACM Transactions on Computer Systems*, 23(1):2–50, 2005.

[17] F. McCown, S. Chan, M. L. Nelson, and J. Bollen. The availability and persistance of web references in D-Lib Magazine. In *5th International Web Archiving Workshop (IWAW'05)*, September 2005.

[18] F. McCown and M. L. Nelson. Evaluation of crawler policies for a web-repository crawler. In *HYPERTEXT '06: Proceedings of the seventeenth ACM conference on Hypertext and hypermedia*, 2006.

[19] M. L. Nelson and B. D. Allen. Object persistence and availability in digital libraries. *D-Lib Magazine*, 8(1), 2002.

[20] S. Pandey, S. Roy, C. Olston, J. Cho, and S. Chakrabarti. Shuffling a stacked deck: the case for partially randomized ranking of search engine results. In *VLDB '05: Proceedings of the 31st international conference on very large data bases*, pages 781–792, 2005.

[21] S.-T. Park, D. M. Pennock, C. L. Giles, and R. Krovetz. Analysis of lexical signatures for improving information persistence on the World Wide Web. *ACM Transactions on Information Systems*, 22(4):540–572, 2004.

[22] T. A. Phelps and R. Wilensky. Robust hyperlinks cost just five words each. Technical Report UCB/CSD-00-1091, EECS Department, University of California, Berkeley, 2000.

[23] N. Shivakumar and H. Garcia-Molina. Finding near-replicas of documents and servers on the web. In *WebDB '98: Selected papers from the International Workshop on The World Wide Web and Databases*, pages 204–212, 1999.

[24] J. A. Smith, F. McCown, and M. L. Nelson. Observed web robot behavior on decaying web subsites. *D-Lib Magazine*, 12(2), 2006.

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

[26] K. Sugiyama, K. Hatano, M. Yoshikawa, and S. Uemura. Refinement of TF-IDF schemes for web pages using their hyperlinked neighboring pages. In *HYPERTEXT '03: Proceedings of the fourteenth ACM conference on Hypertext and hypermedia*, pages 198–207, 2003.

[27] H. Van de Sompel and C. Lagoze. Notes from the interoperability front: A progress report on the Open Archives Initiative. In *ECDL '02: Proceedings of the 6th European Conference on Research and Advanced Technology for Digital Libraries*, pages 144–157, 2002.

[28] H. Van de Sompel, J. A. Young, and T. B. Hickey. Using the OAI-PMH ... differently. *D-Lib Magazine*, 9(7/8), 2003.