# OPAL: IN VIVO BASED PRESERVATION FRAMEWORK FOR LOCATING LOST WEB PAGES

by

Terry L. Harrison
B.S. May 1992, James Madison University

A Thesis Submitted to the Faculty of
Old Dominion University in Partial Fulfillment of the
Requirement for the Degree of

MASTER OF SCIENCE

COMPUTER SCIENCE

OLD DOMINION UNIVERSITY
August 2005

Approved by:

_____
Michael L. Nelson (Director)

_____
Johan Bollen (Member)

_____
Ravi Mukkamula (Member)

## ABSTRACT

## OPAL: IN VIVO BASED PRESERVATION FRAMEWORK FOR LOCATING LOST WEB PAGES

Terry L. Harrison
Old Dominion University, 2005
Director: Dr. Michael L. Nelson

We present Opal, a 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 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. Using the OAI-PMH to facilitate inter-Opal learning extends the utilization of OAI-PMH in a novel manner. 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.

## ACKNOWLEDGMENTS

I would like to thank Dr. Michael Nelson for his superb guidance and patience throughout this period. I would also like to recognize the other members of my Thesis committee for their support. Thanks to Dr. Bollen and to Dr. Mukkamula.

I acknowledge that I should acknowledge many people, those who brought me food when I was hungry, took care of my essential needs while I endured the trials and tribulations of this beast, and my friends that chatted with me online, late at night, to help keep my sanity intact. Of course, I would like to thank my parents. We are the champions, my friends, and we will keep on fighting, until the end; but no need to go on and on.

# TABLE OF CONTENTS

Page

Chapter Page

# LIST OF TABLES

# LIST OF FIGURES

Figure Page

Figure Page

Figure Page

# CHAPTER I

# INTRODUCTION

Digital preservation projects are striving to find balance between refreshing, migration, and emulation. Refreshing is the copying of bits to different systems and migration is the transferring of data to newer system environments (RLG, 1996). Rothenberg (1999) discusses emulation, replicating the functionality of an obsolete system. Digital preservation projects typically involve controlled environments and collections and typically do not 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 Archives' "Wayback Machine") and research projects (e.g. CiteSeer and NSDL).

We propose preservation which relies on the "living" web as "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". Fig. 1 through Fig. 4 show the WI refreshing and migrating web documents, frequently as side-effects of de-duping efforts and user services. We define a framework for harnessing the collective behavior of WI to locate web pages that are no longer present (http error 404).

Google, Yahoo, MSN, etc. are in a race to acquire more content (CrossRef, 2004;

---

The journal model for this thesis is Information Processing and Management (0306-4573).

Yahoo, 2004), so we fully expect the WI to continue growing in breadth and depth. However, we are less concerned about the performance of any single member of the WI (they may come and go), than with the aggregate performance of the WI. The purpose of this research is to interact with the WI to address the problem of missing web pages.


Fig. 1. Google finds 12 versions of NASA report.

## I.1 MOTIVATION

If you wanted to contact a colleague who had recently left one university for another, you might find that your bookmarked URL results in a 404 error. Searching, you find a copy of the site in Google's cache, but this will not reveal the new telephone number of the colleague, which is present on a relocated version of the site. Opal seeks to bridge this gap by finding the relocated page, using a lexical signature derived from the cached copy, and in such a manner, preserve access to the digital object.

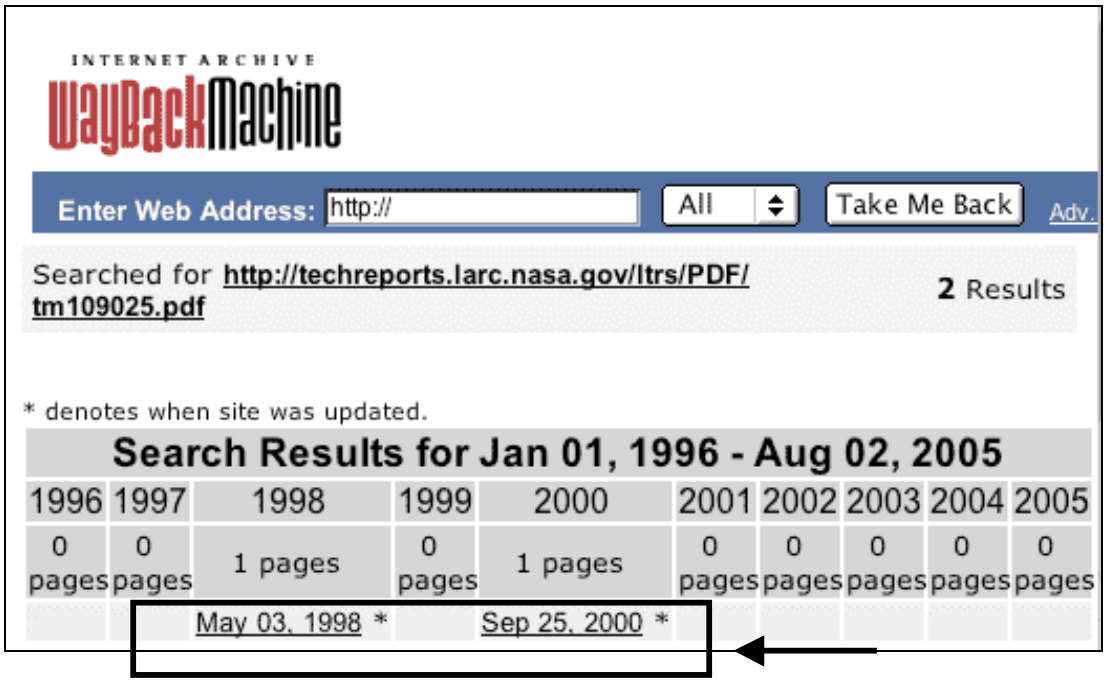



Fig. 2. Wayback Machine finds 2 copies of NASA report.

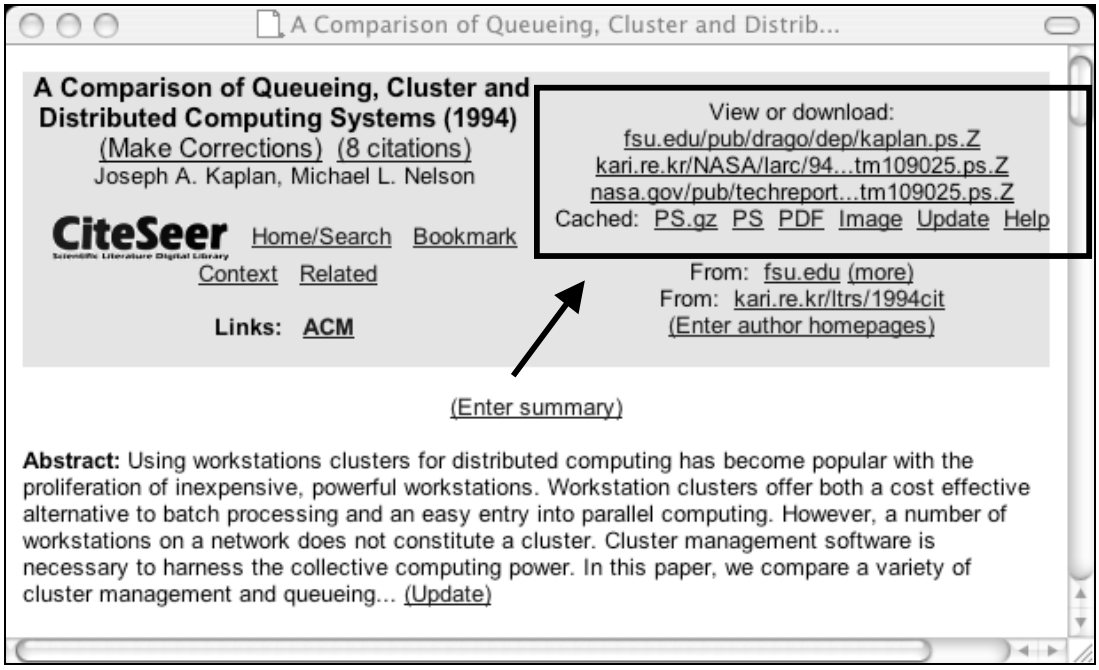



Fig. 3. CiteSeer finds 3 remote and 4 cached versions of NASA report.


Fig. 4. Yahoo finds 3 copies of NASA report.

Other motivating factors of this research are:

- There are no known digital preservation projects which harness the collective services offered though the WI for preservation.
- Difficulties in amassing knowledge about term and document presence on the web may be distributed among cooperating servers which could make the task more feasible.

- There are no known applications of the Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH) as a communications vehicle for machine to machine learning.
- When users are looking for a particular web-page, their evaluation of similar pages can be harnessed to promote the ranking of plausible candidates.

## I.2 AIMS

Opal is a series of cooperating servers that will find new copies of web documents after they are lost by utilizing WI and lexical signatures (LSs). Opal should be modular and extensible. It should demonstrate successful page retrieval using lexical signatures and should be developed to permit the future incorporation of other LS types. Initial Opal design will focus on text based content only. An Opal server will learn from its experiences and be able to share its knowledge with other Opal servers. Servers should be able to learn selectively and such learning from other Opal servers should result in faster collective learning overall.

## I.3 METHODOLOGY

Components for an Opal server were developed and utilized for locating pages, both still on the web and those whose URLs returned 404 errors. Opal searches for cached copies on the WI, based on the 404 URL and creates lexical signatures from them with which to search for similar versions of the page. Term evaluation data learned during lexical signature generation, resulting page matches, and user voting patterns on which pages are most similar are recorded by the Opal server and increase the efficiency and

effectiveness of future requests. Knowledge attained by an Opal server is available for harvesting by others, using the OAI-PMH.

# CHAPTER II

# BACKGROUND

## II.1 THE 404 PROBLEM

Despite well-known guidelines for creating durable URLs (Berners-Lee, 1998), 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 (Kahle, 1997). Koehler (1999; 2004) 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. (2001) 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 (2003) 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. Chan et al. (2005) focused on articles in D-Lib Magazine found an 11 year half-life. Nelson and Allen (2002) 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.

## II.2 LEXICAL SIGNATURES

A lexical signature (LS) is a small set of words that capture the "aboutness" of a document. Phelps & Wilensky (2000) 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 &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 Terry Harrisons in the world, there are far fewer 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 LS 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. These assumptions prevented wide spread adoption of LSs in URLs.

Park et al. (2004) expanded on the work of Phelps & Wilensky, studying the performance of 9 different LS generation algorithms (and retaining the 5-term precedent set by Phelps & Wilensky). 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 fed into Google, results in 117000 pages to select from. On the other hand, the second LS incorporates IDF weighting in addition to TF, and when submitted to Google only results in three pages.

Table 1
Lexical signatures

| Lexical Signature | Calculation Technique | Results from Google |
|---|---|---|
| 2004+terry+digital+harrison+2003 | TF Based | 117000 |
| terry+harrison+thesis+jcdl+awarded | TFIDF Based | 3 |

Both result sets include the author's homepage, however the second LS is more precise. Opal could be enhanced to calculate the LS based on what the user is looking for, the exact page or similar pages. It is worth noting that the second LS would need to be updated if it had contained a zip code which later changed. 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.

## II.3 TFIDF

To evaluate the presence of a given term in a document it could simply be considered as present (1) or not (0). While this may work well in systems working with small controlled vocabularies, it does not take into account term frequency or global weights which may further evaluate the importance of a term, in terms of document relevance and ranking For this we seek a means to add appropriate weight to a term that appears more frequently in a document, bearing in mind how prevalent the term is in the collection as a whole. For this, we turn our discussion to Term Frequency and Inverse Document Frequency (TFIDF) term weighing (Salton & Buckley, 1988).

Term frequency (TF) refers to how often a word appears within a document. Term frequency in a document is typically evaluated as the frequency of a term in a given document normalized by the frequency of the most prevalent term in the document. The assumption is that the more a term appears in a document, the greater the connection between the term and the document. If a one term query is issued against two documents, where one contains two instances of the term and the second contains 10 instances of it, then according to term frequency, if this term is the most frequent in both, then the second document would be more relevant. Should a term appear with high frequency in most documents within a collection, then they will likely all be returned, which would reduce the precision of the result set. Another issue is determining how to evaluate the importance of one query term versus any other. For example, the word “the” appears in far more documents than “hypochondriac”, but few user needs would be fulfilled by returning the document containing the greatest frequency of “the”.

Terms that occur less frequently in a collection of documents have a more unique

connection with those documents in which they do appear. Likewise, the greater the number of documents in a collection that contain a term, the weaker it is at defining a particular document. For example, "NASA" would be a rare term if the collection consisted of newspaper articles. But if the collection was one of NASA documents, "NASA" would occur so frequently as to be meaningless. These considerations are taken into account through the use of the Inverse Document Frequency (IDF). The IDF of a term is evaluated at a collection level by taking the log of the total number of documents (N) over the number of documents in which the term appeared (n), such that a term's IDF varies inversely with N/n (Salton & Buckley, 1988).

A popular and effective term-weighing approach is to combine TF and IDF, weighing in both the frequency of a term in a document along with its relative uniqueness in the collection. This methodology is well suited for long documents (where term frequency alone may over/underestimate document relevance and ranking) and static collections, as the addition of documents necessitates the re-evaluation of term IDF. The TFIDF methodology may also be applied to queries, utilizing normalized query term frequency and the term IDF derived from the collection.

Through the use of local TF and global IDF term weights, TFIDF acts as a performance compromise between the precision and recall evaluations of document relevance and subsequently ranking. Rather than rank documents with the greatest number of matches (to the query) of key terms, instead both the query and documents are processed to take into account the uniqueness of terms in the scope of the collection, as well as each term's frequency in a given document (or query). This resolves the earlier issue involving a query containing the term "the". IDF factors in the significance of the

term in determination of relevance/ranking. Likewise, multiple instances in a document of a very unique term to the collection would give the document an increased relevance to a query which includes the term. Using TFIDF, a document containing a very unique term that matches the query could be ranked higher than another document which contains several matching terms of lesser significance.

### II.4 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. The OAI-PMH initially released by the Open Archives Initiative (OAI) in January of 2001 (Lagoze & Van De Sompel, 2001) and later enhanced with a 2.0 version release in June of 2002 (Van De Sompel & Lagoze, 2002). 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 of 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 (Nelson, 2001).

The OAI-PMH data model can briefly be described in terms of the repository, resource, item, record, datestamp, metadata format, and set. A repository is the home of a collection of publications. The term "resource" refers to an object, be it a digital one, like a scholarly publication or, as we mentioned earlier, a real world object like a stature. The "item" for a resource refers to all of its metadata, in all of the different metadata formats in which it is available. This could be a Dublin Core (required) and an RFC-1807 record for a given resource. "Record" refers to a single metadata format description for a resource. The metadata format is a property, which distinguishes one record for a resource from another record for the same resource. "Datestamp" is used to describe the date of entry of a record or its most recent update (whichever is the most recent). The "metadata format" is the definition of which bibliographic format (such as MARC or RFC-1807) is used to format the record. A record can be available in several metadata formats. The use of "set" is typically used to describe the subject nature of a record (i.e. biology) but has also been used to define the source of origin for the record (as it may be

passed along from aggregator to aggregator) (Lagoze, C. & Van De Sompel, H., 2001).

Use of OAI-PMH can benefit areas where the update and synchronization of distributed metadata is desired. Metadata can represent more than just publications. Objects, real or digital, or even raw data sets could be represented with metadata. New metadata formats can be created that may more accurately represent a different type of object rather than trying to force a metadata mapping onto an ill-suited pre-existing format. OAI-PMH compliance requires support of the Dublin Core metadata format, but since all fields in this format are optional, conforming to this requirement does not require much additional overhead. Van De Sompel discusses usage of OAI-PMH for thesauruses, usage logs, and as an OpenURL registry (Van De Sompel et al., 2003).

## II.5 WEB CRAWLERS AND CACHES

Search engines, like Google and Yahoo, use web crawlers (or spiders) are collect web site information across the WWW. Given a starting location, it caches the site (or some portion of it) and follows the associated hyperlinks recursively across the network to new pages. Though the WWW is a distributed collection, the analogy to a spider walking its web is a fairly good one. Architecturally speaking, the crawler must store the URLs it finds and keep track of its progress across them. It must also store the pages that it collects. While some cache only portions of the site (what it can get in a fixed amount of time, or a predetermined amount of data), others attempt to cache the entire site. The basic purpose of the crawl is to provide data to discover resources. Once gathered, this data can be processed and indexed.

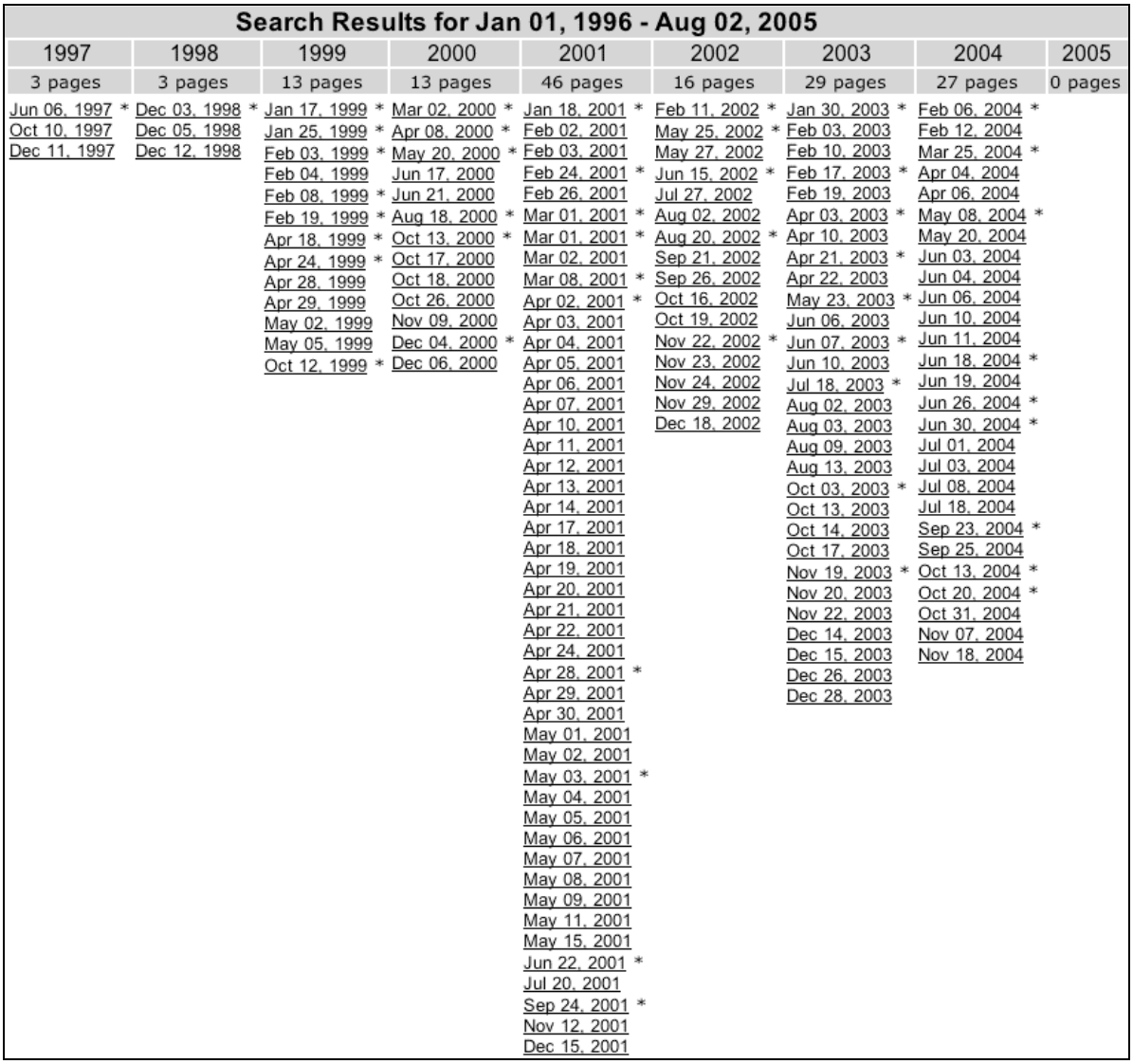
The WWW is growing at a phenomenal rate. Google, Yahoo, and MSN claim to

have indexed 8,4, and 5 billion pages, respectively, and the current estimated size of the Web that can be reached by crawlers is estimated at 11.5 billion (Gulli & Signorini, 2005). Entry point URLs are carefully chosen to promote more efficient traversal of the web. Communications issues abound, as the crawler must haul ingest amounts of data over limited bandwidth. Not all sites wish to be crawled and crawlers should be respectful of any robot.txt (Mauldin & Schwartz, 1996) protocol messages that may be in place. In addition, Web sites are far from static, popping in and out of existence and often undergoing revision. Crawlers must consider the freshness of their cached sites and consider the scheduling of the crawlers revisit for site verification and periodic update (Cho & Garcia-Molina, 2003).

### II.5.1 Internet Archive

Web pages change over time, but only a single (if any) cached version is available via commercial search engines. When web page content is replaced and then subsequently cached by a search engine, the older version of the cache is no longer available. In 1996, Brewster Kahle established the Internet Archive (IA) to prevent web content from disappearing in scenarios like these (Kahle, 1997). Since then, IA has been crawling the web and maintains over 40 billion cached copies of web pages as they exist over time (Internet Archive, 2005). The number of cached versions for a URL depend on the number of times it has been crawled by the IA. Fig. 5 shows the cached IA crawls available for the Old Dominion University Computer Science homepage (http://www.cs.odu.edu), via its online searchable interface, “The Wayback Machine”. Over 100 cached versions of the ODU page are available from the Wayback interface.

When a page is no longer available from its web server or from the caches of search engines, IA offers a potential source for reviving the dead link.



**Search Results for Jan 01, 1996 - Aug 02, 2005**

| 1997 | 1998 | 1999 | 2000 | 2001 | 2002 | 2003 | 2004 | 2005 |
|---|---|---|---|---|---|---|---|---|
| 3 pages | 3 pages | 13 pages | 13 pages | 46 pages | 16 pages | 29 pages | 27 pages | 0 pages |
| Jun 06, 1997 * | Dec 03, 1998 * | Jan 17, 1999 * | Mar 02, 2000 * | Jan 18, 2001 * | Feb 11, 2002 * | Jan 30, 2003 * | Feb 06, 2004 * | |
| Oct 10, 1997 | Dec 05, 1998 | Jan 25, 1999 * | Apr 08, 2000 * | Feb 02, 2001 | May 25, 2002 * | Feb 03, 2003 | Feb 12, 2004 | |
| Dec 11, 1997 | Dec 12, 1998 | Feb 03, 1999 * | May 20, 2000 * | Feb 03, 2001 | May 27, 2002 | Feb 10, 2003 | Mar 25, 2004 * | |
| | | Feb 04, 1999 | Jun 17, 2000 | Feb 24, 2001 * | Jun 15, 2002 * | Feb 17, 2003 * | Apr 04, 2004 | |
| | | Feb 08, 1999 * | Jun 21, 2000 | Feb 26, 2001 | Jul 27, 2002 | Feb 19, 2003 | Apr 06, 2004 | |
| | | Feb 19, 1999 * | Aug 18, 2000 * | Mar 01, 2001 * | Aug 02, 2002 | Apr 03, 2003 * | May 08, 2004 * | |
| | | Apr 18, 1999 * | Oct 13, 2000 * | Mar 01, 2001 * | Aug 20, 2002 * | Apr 10, 2003 | May 20, 2004 | |
| | | Apr 24, 1999 * | Oct 17, 2000 | Mar 02, 2001 | Sep 21, 2002 | Apr 21, 2003 * | Jun 03, 2004 | |
| | | Apr 28, 1999 | Oct 18, 2000 | Mar 08, 2001 * | Sep 26, 2002 | Apr 22, 2003 | Jun 04, 2004 | |
| | | Apr 29, 1999 | Oct 26, 2000 | Apr 02, 2001 * | Oct 16, 2002 | May 23, 2003 * | Jun 06, 2004 | |
| | | May 02, 1999 | Nov 09, 2000 | Apr 03, 2001 | Oct 19, 2002 | Jun 06, 2003 | Jun 10, 2004 | |
| | | May 05, 1999 | Dec 04, 2000 * | Apr 04, 2001 | Nov 22, 2002 * | Jun 07, 2003 * | Jun 11, 2004 | |
| | | Oct 12, 1999 * | Dec 06, 2000 | Apr 05, 2001 | Nov 23, 2002 | Jun 10, 2003 | Jun 18, 2004 * | |
| | | | | Apr 06, 2001 | Nov 24, 2002 | Jul 18, 2003 * | Jun 19, 2004 | |
| | | | | Apr 07, 2001 | Nov 29, 2002 | Aug 02, 2003 | Jun 26, 2004 * | |
| | | | | Apr 10, 2001 | Dec 18, 2002 | Aug 03, 2003 | Jun 30, 2004 * | |
| | | | | Apr 11, 2001 | | Aug 09, 2003 | Jul 01, 2004 | |
| | | | | Apr 12, 2001 | | Aug 13, 2003 | Jul 03, 2004 | |
| | | | | Apr 13, 2001 | | Oct 03, 2003 * | Jul 08, 2004 | |
| | | | | Apr 14, 2001 | | Oct 13, 2003 | Jul 18, 2004 | |
| | | | | Apr 17, 2001 | | Oct 14, 2003 | Sep 23, 2004 * | |
| | | | | Apr 18, 2001 | | Oct 17, 2003 | Sep 25, 2004 | |
| | | | | Apr 19, 2001 | | Nov 19, 2003 * | Oct 13, 2004 * | |
| | | | | Apr 20, 2001 | | Nov 20, 2003 | Oct 20, 2004 * | |
| | | | | Apr 21, 2001 | | Nov 22, 2003 | Oct 31, 2004 | |
| | | | | Apr 22, 2001 | | Dec 14, 2003 | Nov 07, 2004 | |
| | | | | Apr 24, 2001 | | Dec 15, 2003 | Nov 18, 2004 | |
| | | | | Apr 28, 2001 * | | Dec 26, 2003 | | |
| | | | | Apr 29, 2001 | | Dec 28, 2003 | | |
| | | | | Apr 30, 2001 | | | | |
| | | | | May 01, 2001 | | | | |
| | | | | May 02, 2001 | | | | |
| | | | | May 03, 2001 * | | | | |
| | | | | May 04, 2001 | | | | |
| | | | | May 05, 2001 | | | | |
| | | | | May 06, 2001 | | | | |
| | | | | May 07, 2001 | | | | |
| | | | | May 08, 2001 | | | | |
| | | | | May 09, 2001 | | | | |
| | | | | May 11, 2001 | | | | |
| | | | | May 15, 2001 | | | | |
| | | | | Jun 22, 2001 * | | | | |
| | | | | Jul 20, 2001 | | | | |
| | | | | Sep 24, 2001 * | | | | |
| | | | | Nov 12, 2001 | | | | |
| | | | | Dec 15, 2001 | | | | |

Fig. 5. Dated links to cached copies of http://www.cs.odu.edu.

## II.6 RELATED WORK

### II.6.1 LOCKSS

The LOCKSS ("Lots of Copies Keeps Stuff Safe") project is an elegant approach for distributing content amongst known participants (Reich & Rosenthal, 2001). 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. The cache synchronization algorithm is described by Maniatis et al. (2003), but a short description follows. The caches periodically multicast hashes of the publications they hold. If a participant fails to respond within a randomly chosen time interval, a "sloppy election" is held. This election is performed to identify and isolate where the damage is in the network. If a participant's vote does not match the majority's vote for a particular collection of files, a representative of the majority will transmit a copy of the files in question – if you have voted with the majority in the past. This constraint prevents newcomers from arriving late and claiming they "lost" content they never had.

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.

### II.6.2 Web Document Similarity

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 (Brin et al., 1995; Shivakumar & Garcia-Molina, 1996).

Fingerprinting has also been used to identify duplicate and near-duplicate web pages for the purpose of optimizing web crawler performance (Shivakumar & Garcia-Molina,1998; Cho, Shivakumar, & Garcia-Molina, 2000). 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.

### II.6.3 404 Services

PERIDOT is a web tool that automatically maps and stores features of web pages ahead of time, to use later to detect changes. PERIDOT identifies page content though a fingerprinting process. Site owners must sign up for the service or have it installed

locally to begin analyzing and monitoring pages (Twist, 2004). This requires that the system be in place before the pages disappear. Instead of these a priori installation requirements, Opal can retrieve online caches that were made before an Opal installation.

Linkguard is web service company with a 40TB database that "logs all the links on the web" (Ward, 2000). Clients that sign up for the service are notified if and when in-bound links to their site are broken. This could happen if the client moves a page and all previously working in-bound links become broken or if another page, with in-bound links to the client page, moves (or goes away). While this is an interesting service, Linkguard does not help to locate the missing page.

Walden Path Manager (WPM) is a tool that evaluates changes in web pages and evaluates the suitability of replacement candidates. The goal is slightly different than Opal, and 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 X to X'. 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 (Dalal, 2004).

# CHAPTER III

# OPAL

## III.1 INTRODUCTION

Although Phelps & 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, say $O_1$ .. $O_n$. We then take advantage of the fact that most web servers allow for custom 404 error pages to be returned. This is accomplished on Apache servers by appending the configuration file, httpd.conf with the name of the document to handle the 404 error code (Fig. 6). The goal is to make it as easy as possible for web server administrator to take advantage of Opal servers. By adding a JavaScript page tag to this custom 404 page (Fig. 7), 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 a web server administrator will have to perform to use Opal. 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.

```
#ErrorDocument 500 "The server made a boo boo."
#ErrorDocument 402 http://www.example.com/subscription_info.html
 ErrorDocument 404 /404.html
```

Fig. 6. Configuring Apache's httpd.conf file to return a custom 404 page.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
                      "http://www.w3.org/TR/html4/loose.dtd">
<html lang="en">
<head>
        <meta http-equiv="content-type" content="text/html; charset=utf-8">
        <title>404 JavaScript Page</title>
</head>
<body bgcolor="beige">

<script language="javascript" type="text/javascript">
<!-- Hide script from old browsers
    window.location="http://www.cs.odu.edu/~tharriso/thesis/code/opal1.pl?new_url="+ document.URL +"&page=2"
// end hiding script from old browsers -->
</script>

</body>
</html>
```

Fig. 7. JavaScript redirect page.

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.

## III.2 DESIGN CONSIDERATIONS

### III.2.1 Discovery

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. Fig. 8 shows an OAI-PMH Identify response listing "friends" of Opal1.com.

### III.2.2 Freshness

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

```
<?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>
```

Fig. 8. Opal OAI-PMH Identify response.

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 thought 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.

### III.3 ARCHITECTURE

Fig. 9 depicts the Opal server architecture. When a user requests a web page (URL) from a web server (Step 1) and that page is not found (404), a custom 404 page is sent back to the client (2). This page contains a redirect which passes the 404 URL to the Opal Server (3). At this point, (4) Opal searches externally for the existence of cached versions of the 404 page, and well as internally for any known lexical signature or affirmed similar URLs. The amount of data presented to the user depends on the interface level (basic or advanced) selected. From the available choices, the user selects how they would like to continue their exploration. If they select the description of one of the resulting potential pages, (5) this selection acts as a vote, indicating a level of human decision that this page is related. Opal records the vote (6), opens a new window and then (7) 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.

### III.4 GRAPHICAL USER INTERFACE

The Opal GUI divides the screen into five components: Welcome, Interface Options, Similar Pages, Caches, and Search Options, as illustrated in Fig. 10.

The "Welcome" section provides a short, textual of the service Opal provides. A page number is also listed in the prototype implementation which shows where the user is

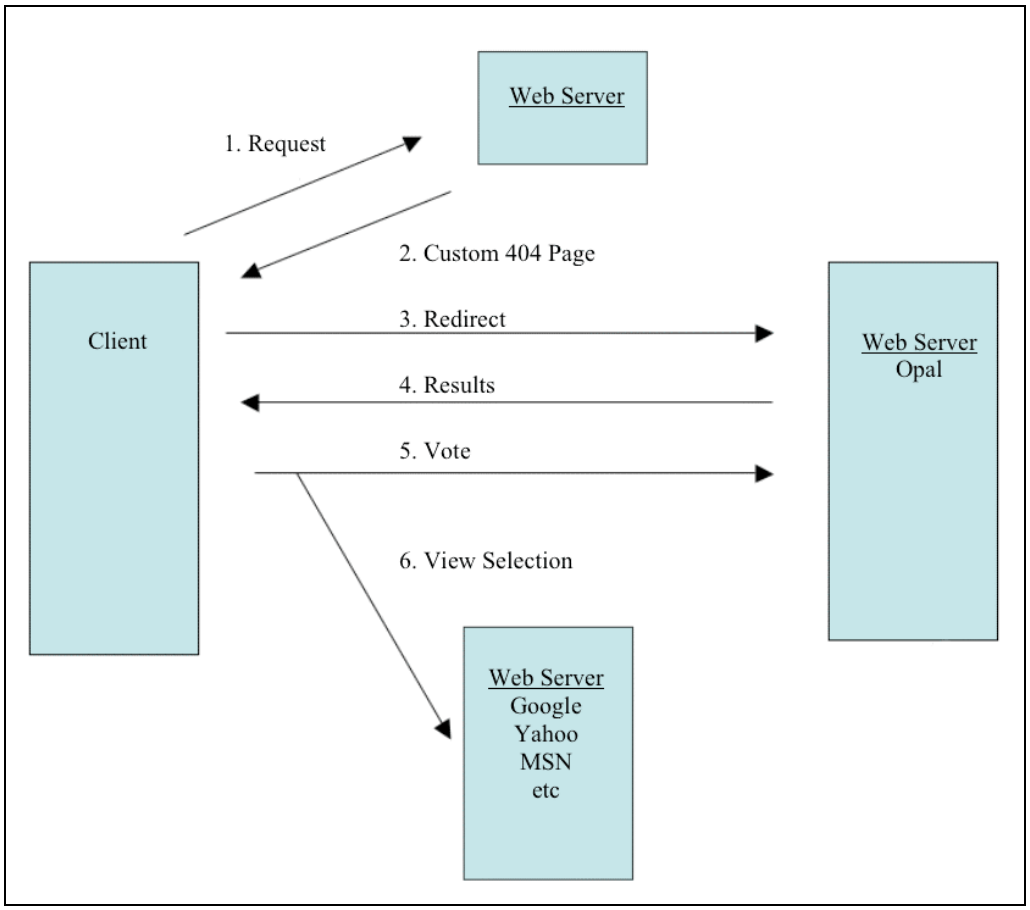



Fig. 9. Opal high level architecture.

in the search process.

The "Interface Options" section displays the 404 URL and indicates the level

(Basic or Advanced) or the GUI being currently used. The user can select to change the GUI level here. The Advanced GUI permits the user to modify or change the 404 URL. Stand-alone Opal systems, start with this URL field blank and permit user input.

The "Similar Pages" section displays brief descriptions of URLs which have previously been determined, (via a vote tally) to be similar to the 404 URL currently in question.

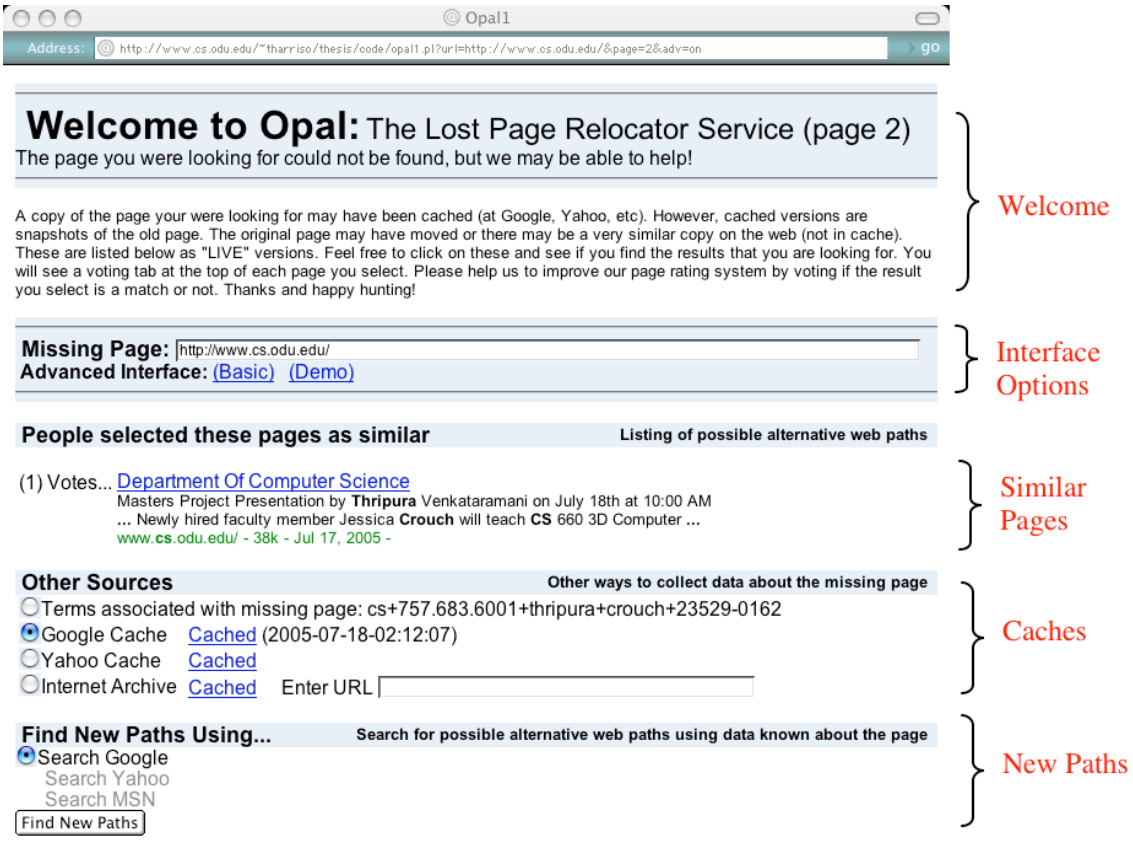



Fig. 10. GUI components.

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. Opal currently looks for the existence of cached copies of the 404 URL in Google in all GUI modes and Yahoo caches in the more advanced modes (Advanced and Demo). In these advanced modes, a link to IA’s Wayback Machine is presented, from which the user can search for and select an IA cached copy to use for lexical signature calculation. 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 you can print out may be entirely satisfactory. For use cases where the live replacement site is required, this provides the user the opportunity to select the most appropriate cached content with which to continue the pursuit.

Depending on the GUI level, the user may be offered to choice in deciding which search engine (Google, Yahoo, MSN, etc.) to use the LS (either manually selected by the user if a matching LS is already known by Opal, or dynamically calculated as part of finding new paths) in to search for possible alternate paths to the content in the 404 URL.

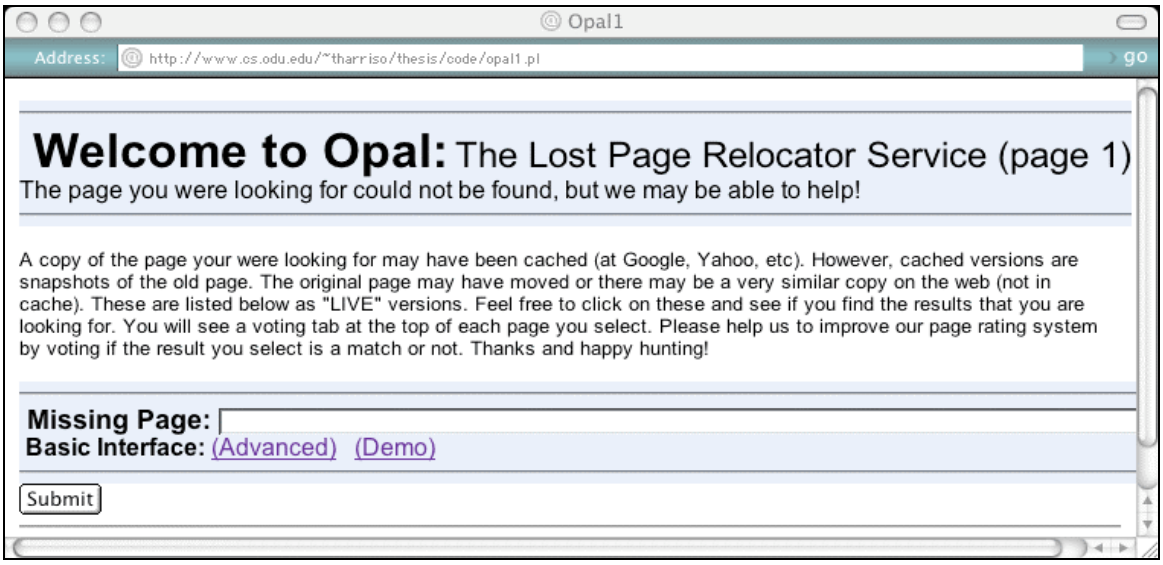



Fig. 11. Basic interface, standalone version.

### III.4.1 Basic Interface

A user may opt to conduct a URL search directly from the base URL of an Opal server. This is the approach taken when Opal is viewed as a standalone system. Opal presents a minimal entry interface denoted as “page 1” in which the user can enter in the 404 URL. The user starts in the Basic interface mode (Fig. 11). The user can either submit a URL or switch to a more advanced mode (explained in their respective sections which follow).

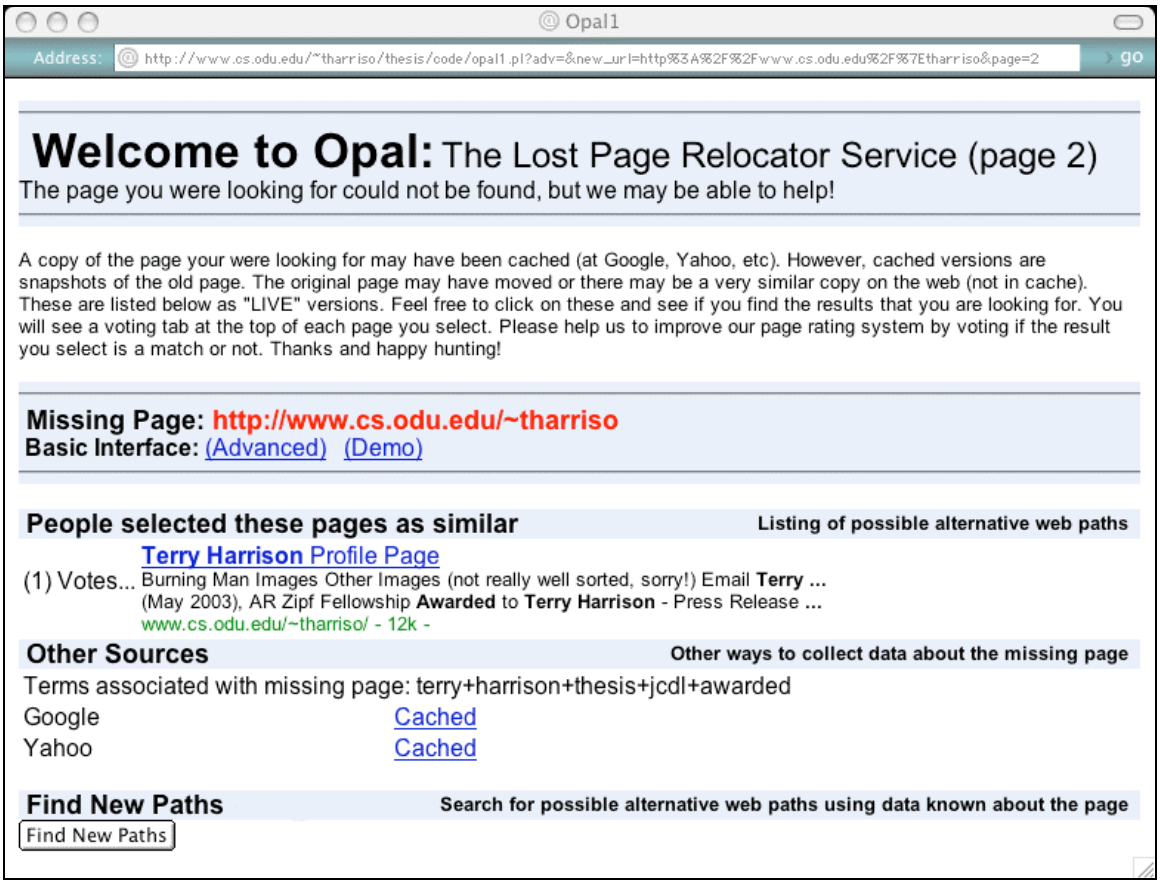



Fig. 12. Basic interface, page 2.

The other entry point to Opal is via a redirect originating from the web server hosting the 404 URL. The name of this URL is passed in, and is displayed. This state, where the 404 URL is known a priori is labeled as “page 2” (Fig. 12). Options are presented which allow the user to engage more advanced GUIs. The user sees that this Opal server has knowledge of a similar page and that it locate both a Google and Yahoo cache copy. If the user clicks on the link to the cache, it will open it in a new window. In the current implementation, the Basic interface only handles Google and Yahoo caches. Additional options are provided in the more advanced interfaces.

After the user submits the request to “Find New Paths”, the GUI will look as before, with two exceptions. First, this new state, which may include a result set, is labeled as “page 3”. In addition, the New Paths section will append any new found URL into a “New Paths Found” section at the bottom of the page (Fig. 13). Here the hyperlinked descriptions of the new found pages will appear and the submit button will be removed. The user can pursue additional options via the more advanced GUIs, if desired.

If no Google or Yahoo cache content has been located and no lexical signature is currently known, the user cannot submit the request (the “Find New Paths” submit button will not appear), but may continue with pursuit of more advanced GUI options.


Fig. 13. Basic interface, page 3, new paths found.

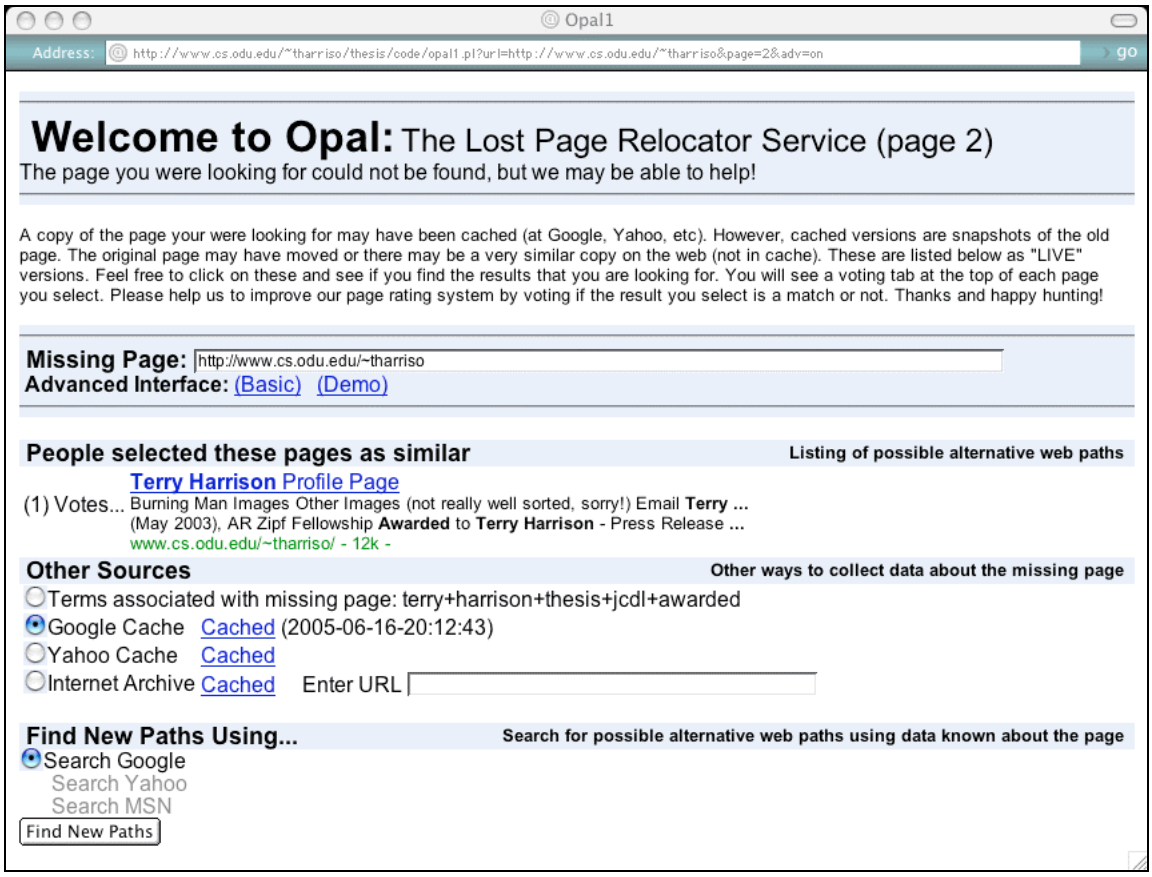



Fig. 14. Advanced interface.

### III.4.2 Advanced Interface

The Advanced GUI (Fig. 14) permits the user to modify the 404 URL to refine searches. Since it is difficult to canonize URLs, http://foo.com/a may be different than http://foo.com/a/. Accordingly, these are considered two different URLs. These variations can be explored in the Missing Page text box (Fig. 15). After URL modification, the user can submit the request, which will update the page with appropriately updated Similar and Source section data. Once the data is synchronized with the modified URL, a subsequent submission will retrieve the new path result set for the new URL.


Fig. 15. Missing page section of the advanced interface.

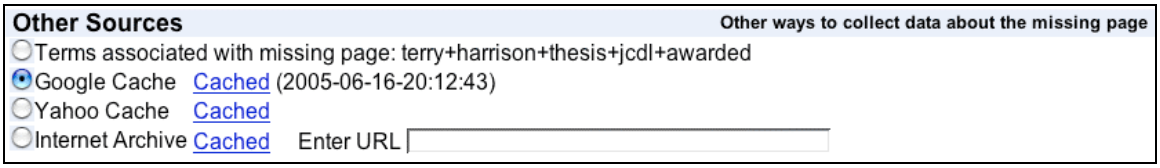



Fig. 16. Source section of the advanced interface.

In the Sources section (Fig. 16), the user is now permitted to select which data set (cached version from a particular cache or a known Lexical Signature) to use to search for new paths. The Internet Archive link takes the user (in a new window) to Internet Archive Wayback Machine cache holdings page for the 404 URL.

At the Wayback Machine web site (Fig. 17), the user can view the various cached copies of the 404 URL held by the Internet Archive. Once the most similar cached copy is located, its Wayback URL can be pasted into the corresponding text box next to the Internet Archive option, and this will become the cache version used to derive a site lexical signature for our search. While many search engines could be selected to search the resulting lexical signature, the prototype only uses Google (Fig. 18).

Results from "Find New Paths" are appended to the screen (Fig. 19), leaving the "Find New Paths" submit button in place. This permits the user to investigate the results set, or modify the search parameters for a new search.

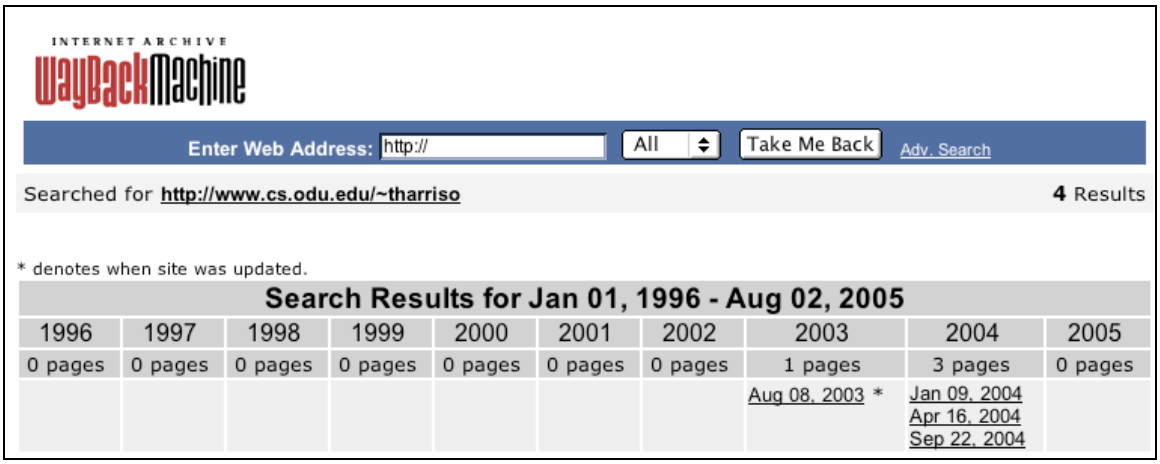



Fig. 17. Wayback Machine search interface.

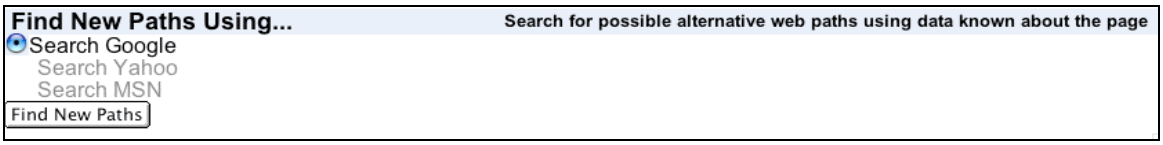



Fig. 18. Choice of search engine to run lexical signature.


Fig. 19. The advanced interface results section.

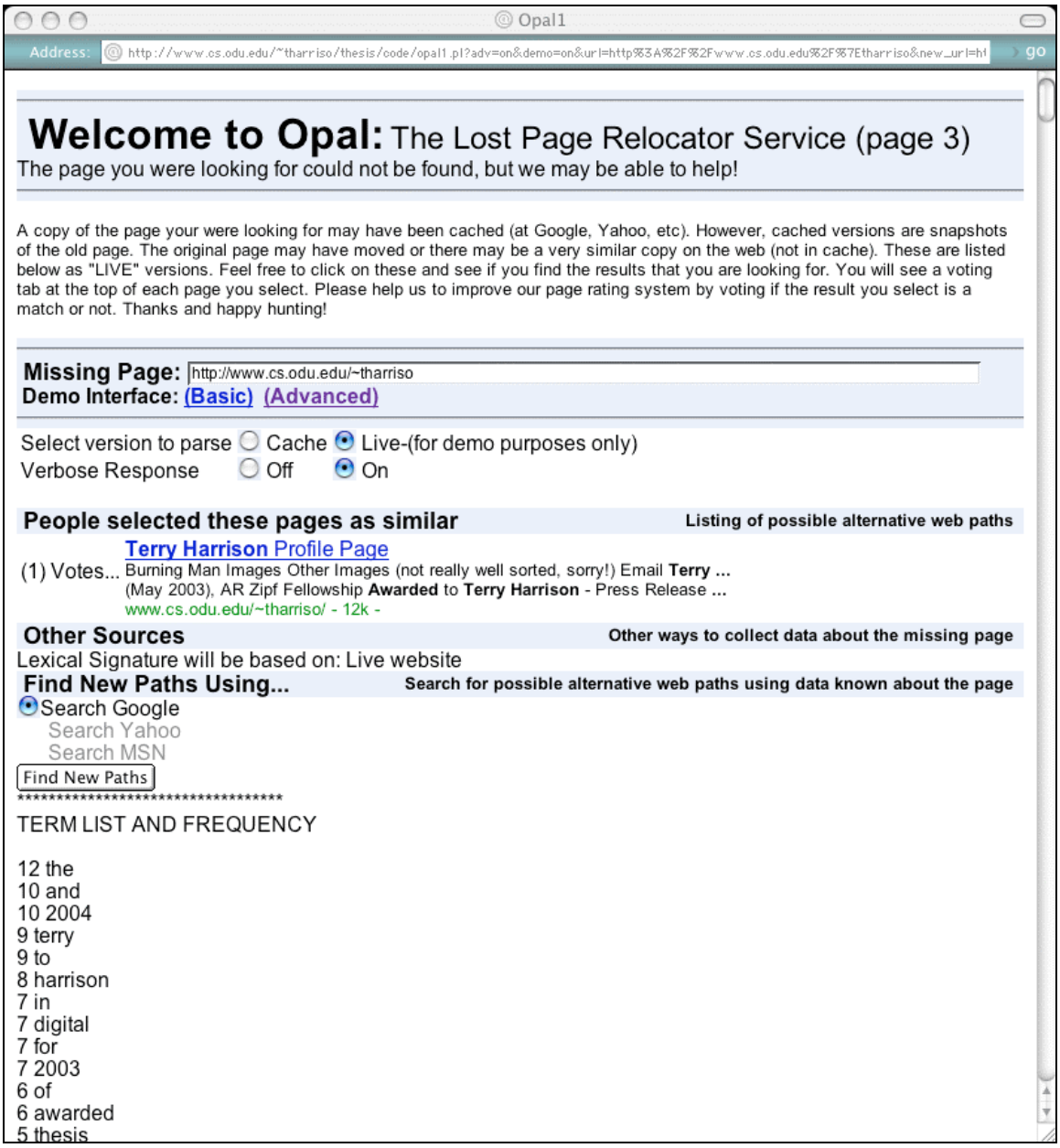



Fig. 20. Demo interface.

### III.4.3 Demo Interface

The demo GUI offers the user some additional features which are currently for demonstration purposes only (Fig. 20). If the user elects to turn Verbose on, then TF and

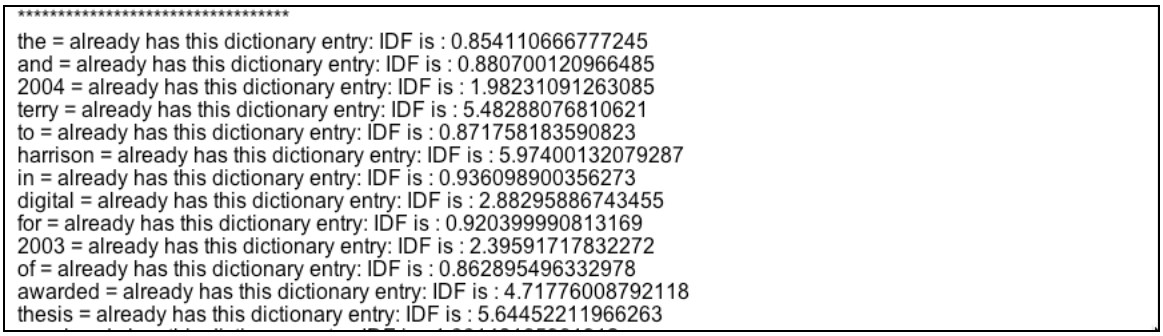



Fig. 21. Demo interface IDF values.

IDF calculation results (Fig. 21) will be printed out along with the results. Any new LS that is calculated is also displayed with the results. While Google is the only search engine implemented our prototype to handle Opal searches with a LS and vote affiliation, users can click on links to see what result sets would be available via other search engines, as shown in Fig. 22, 23 and Fig. 24.

Finally, there is the option for the user to submit a URL which is not 404 in order to compute an LS and conduct searches based on this "live" URL (Fig. 20). This is listed in the demo section because images tags found in live versions can reduce the accuracy of LS creation at this time. Users can manually eliminate image references from the resulting LS after clicking to view the various search engine result sets.

### III.5 OPAL FRAMEWORK

The Opal framework consist one or more collaborating Opal servers. Each Opal server consists of three basic components: the httpd.conf file ErrorDocument to point to a local 404 HTML page, the 404 redirect HTML page, and the Opal script (opal1.pl in the case of our prototype) which responds to the request for the 404 URL. The sever which

contains the Opal script is considered the Opal server. This server contains implementation specific storage structures hold learned data records, including term IDF values and 404 URL voting histories. A set of support scripts permit an Opal server to act as an OAI-PMH compatible 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.

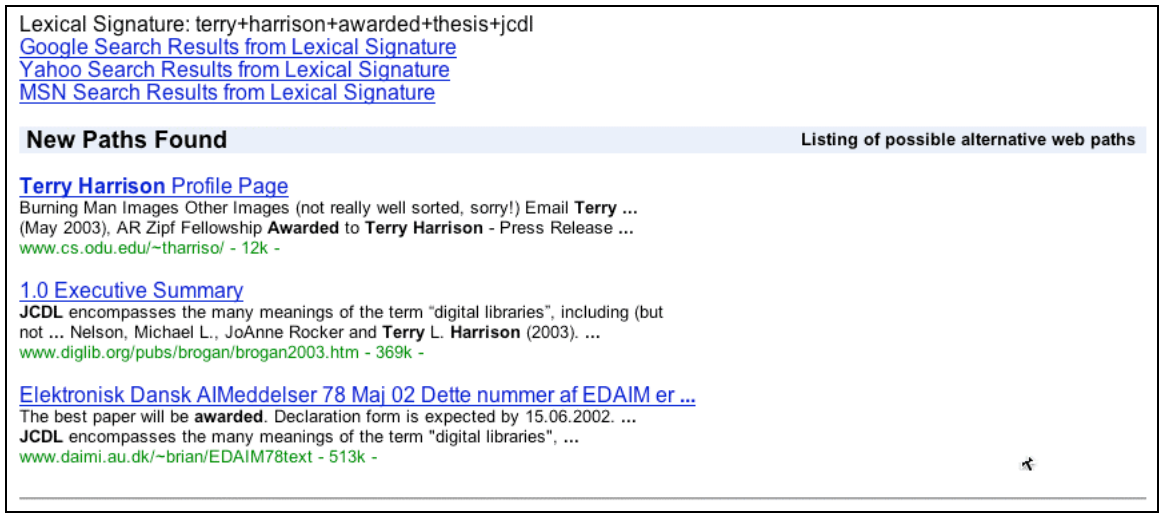



Fig. 22. Demo interface search links and new found pages.

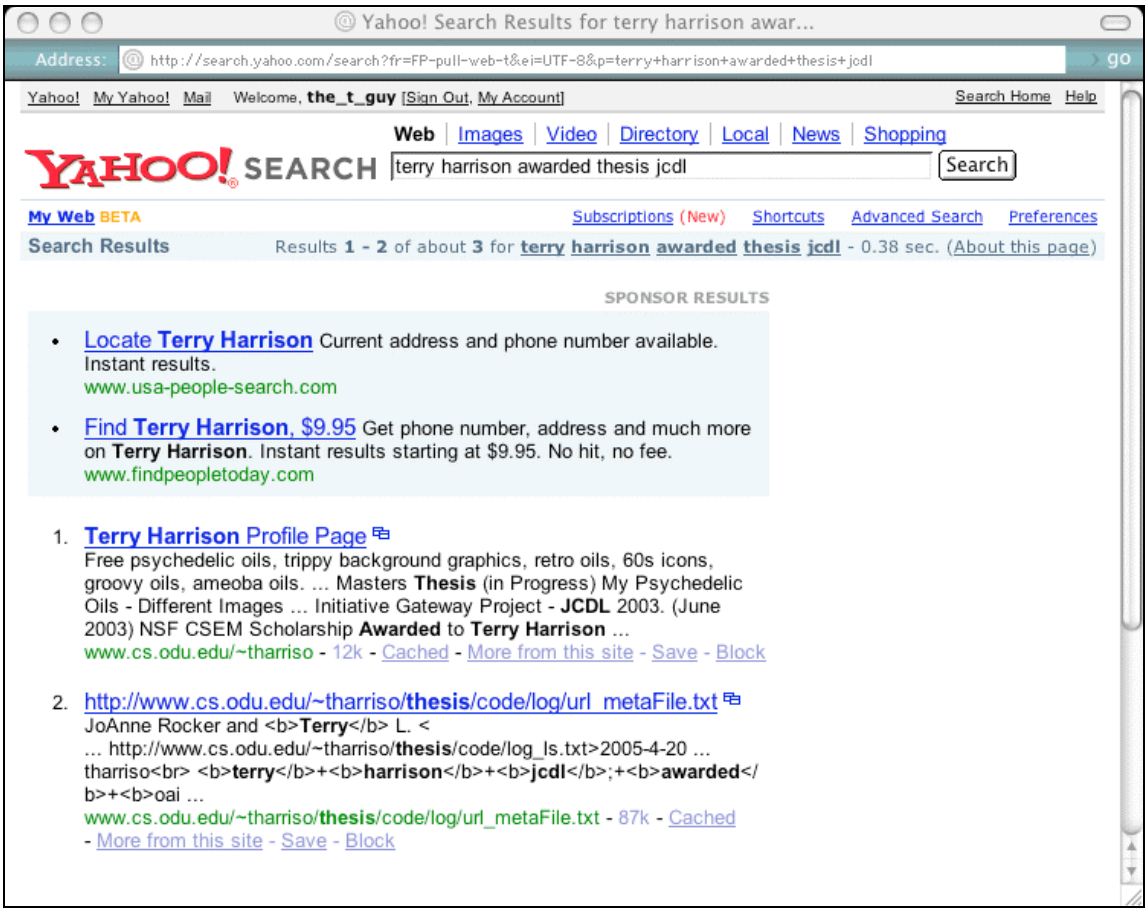



Fig. 23. Demo interface Yahoo search results with lexical signature.


Fig. 24. Demo interface MSN search results with lexical signature.

### III.5.1 Client Requests

Once the an initial request is made, a variety of lookups are conducted to pull together the data needed for the response, as shown in Fig. 25. A Lexical Signature lookup (step 1) 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 , (as determined by the implementation) 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 (step 2), against search engines (Google and Yahoo) to determine if a hyperlink to a cached version can be located (Figs. 26 and 27). 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 (step 3) 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
```

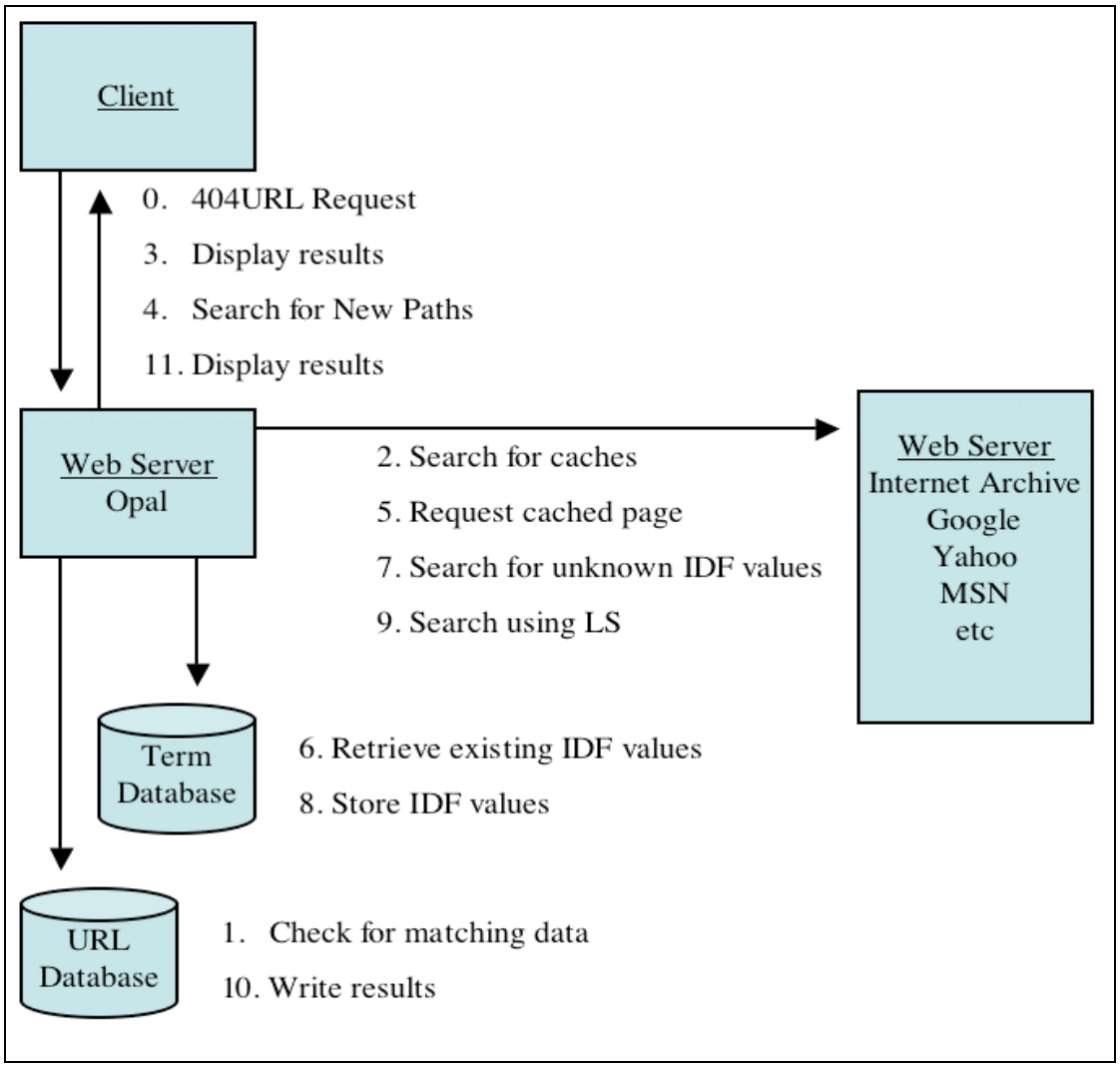



Fig. 25. Handling client requests.

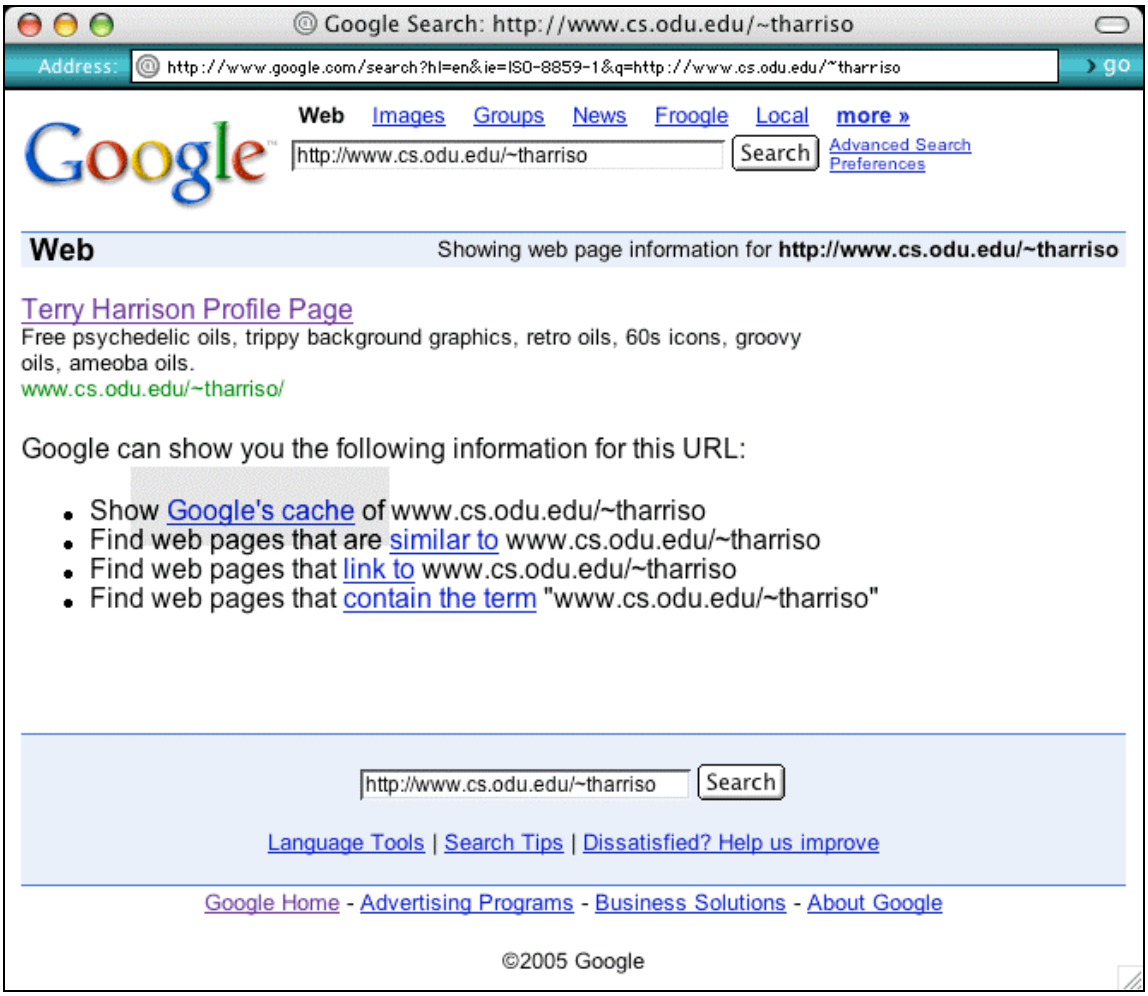



Fig. 26. Locating Google caches.

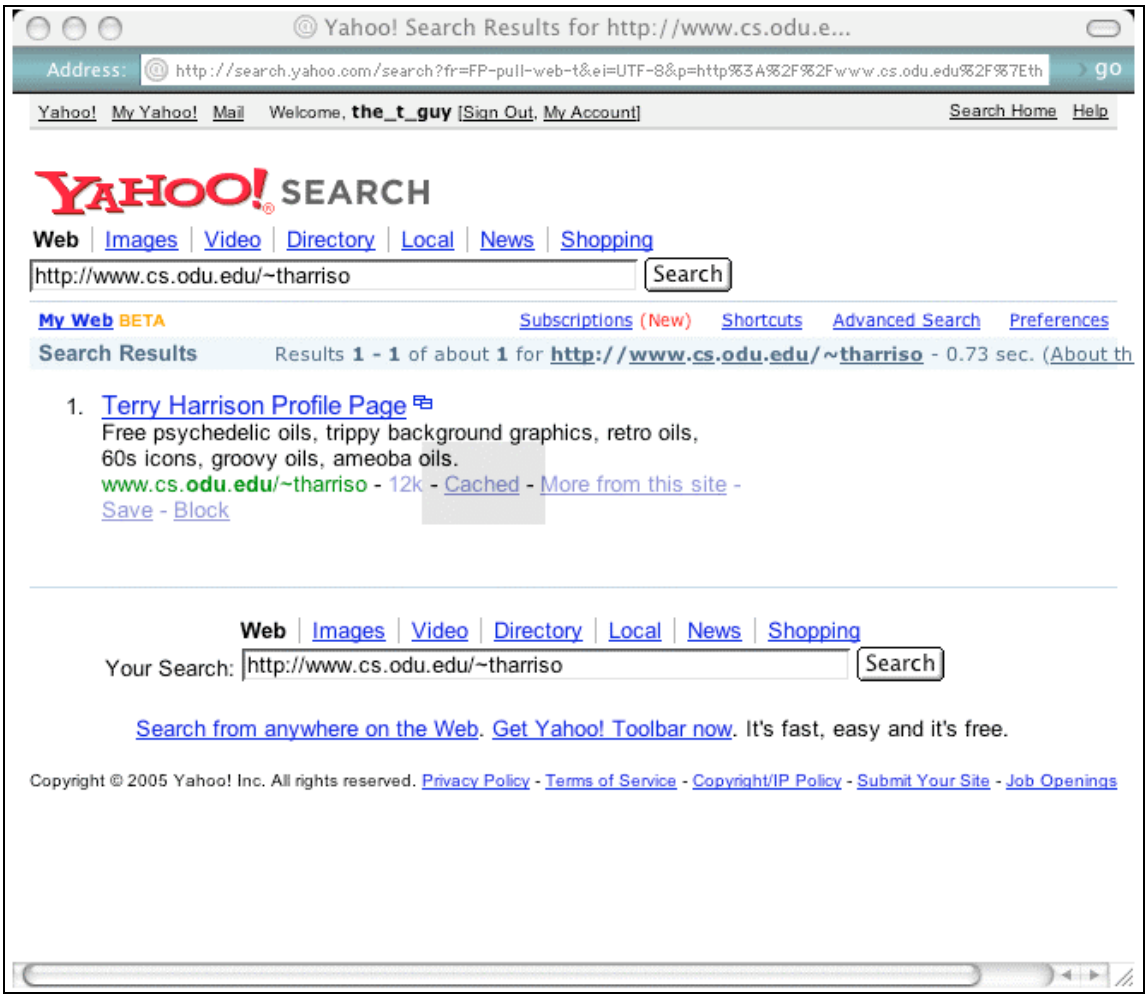



Fig. 27. Locating Yahoo caches.

If the user chooses to explore using a cached version (step 4), then that cache content is retrieved (step 5) and parsed of its terms (step 6). 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 (step 7) to ascertain its value (Fig. 28). This IDF data is appended to the Term Db to reduce the overhead of subsequent lookups (step 8). Term stemming (such as removing prefixes and suffixes) is tempting, to reduce the number of terms requiring lookup, but we do not do this because both term frequency results from Google

as well as search results.

```
the = already has this dictionary entry: IDF is : 0.854110666777245
and = already has this dictionary entry: IDF is : 0.880700120966485
2004 = already has this dictionary entry: IDF is : 1.98231091263085
terry = already has this dictionary entry: IDF is : 5.48288076810621
to = already has this dictionary entry: IDF is : 0.871758183590823
in = already has this dictionary entry: IDF is : 0.936098900356273
digital = already has this dictionary entry: IDF is : 2.88295886743455
harrison = already has this dictionary entry: IDF is : 5.97400132079287
for = already has this dictionary entry: IDF is : 0.920399990813169
2003 = already has this dictionary entry: IDF is : 2.39591717832272
of = already has this dictionary entry: IDF is : 0.862895496332978
thesis = already has this dictionary entry: IDF is : 5.64452211966263
a = already has this dictionary entry: IDF is : 1.00148165961912
awarded = already has this dictionary entry: IDF is : 4.71776008792118
```

Fig. 28. Dictionary IDF entries.

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 (Fig. 29). 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, we believe our estimates will prove more accurate than the techniques employed in previous studies.

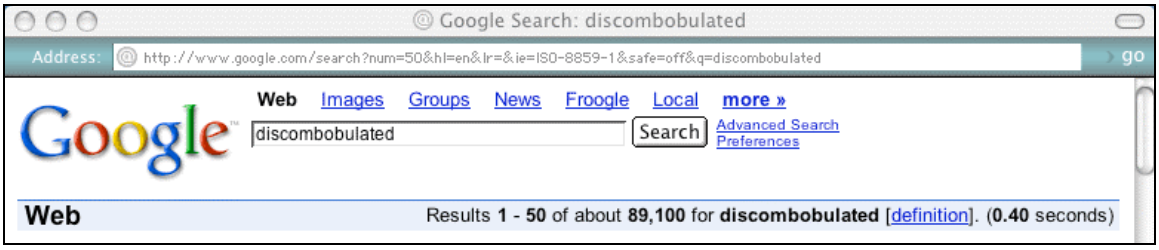



Fig. 29. Gathering IDF data.

Once all TF and IDF values are retrieved, then the standard TFIDF calculation (Phelps & Wilensky, 2000) is performed for each term from the cache and the LS determined (from the top 5 results). The LS is stored in the URL Db and is also used to drive the selected search engine (Fig. 25, step 9) for new paths (as we saw earlier when a pre-existing LS was selected) (Figs. 30 and 31). The resulting URLs are recorded in the URL Db along with their descriptions (Fig 25, step 10), as presented by the search engine.

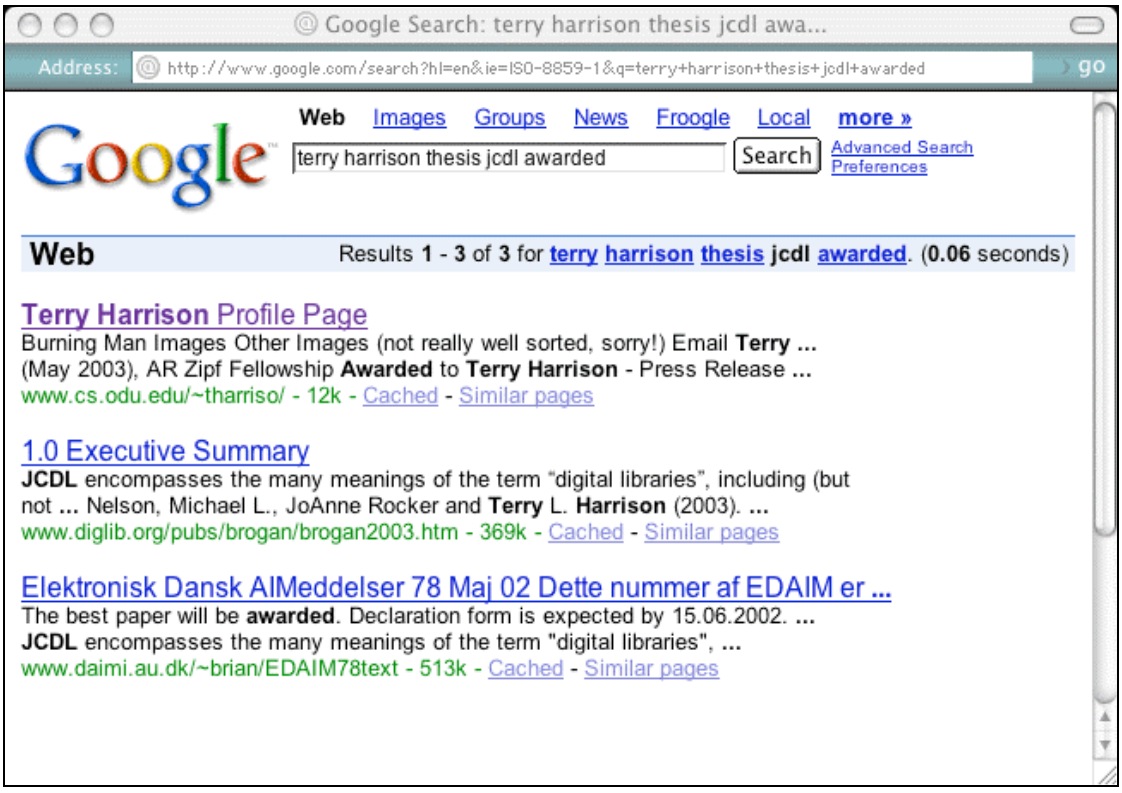



Fig. 30. Searching Google with the lexical signature.

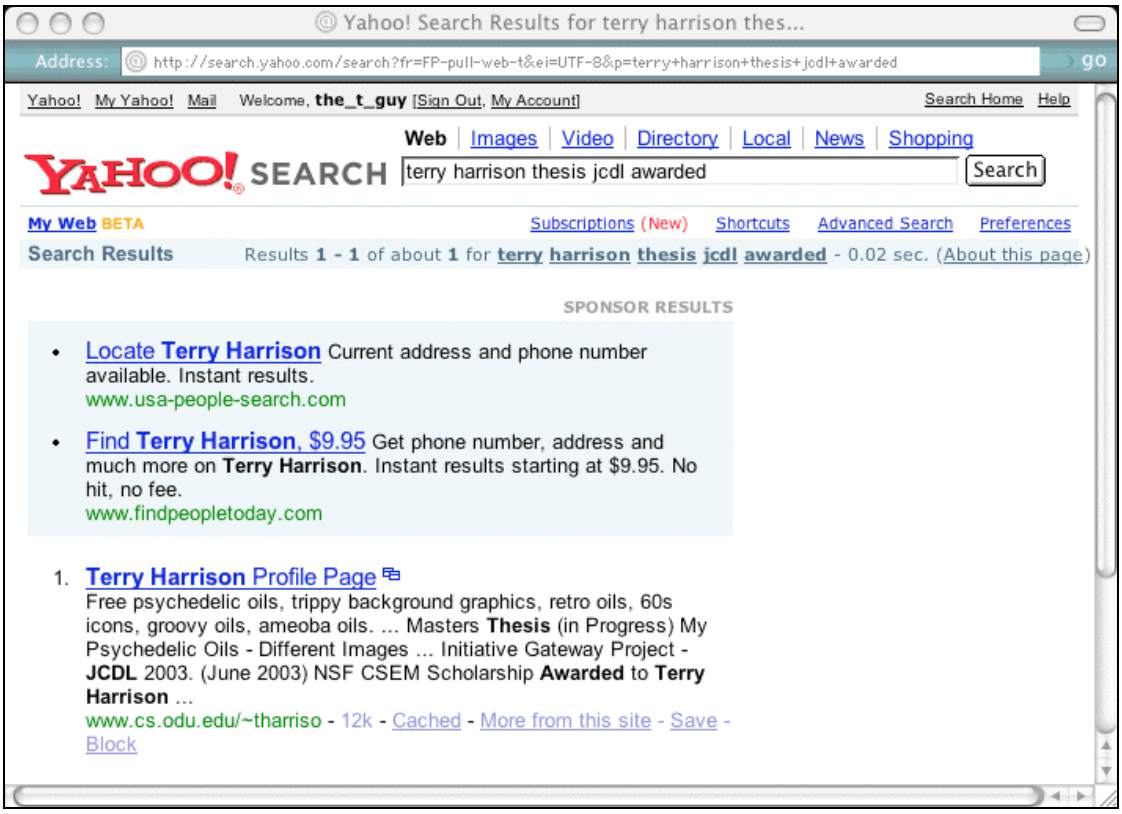



Fig. 31. Searching Yahoo with the lexical signature.

The new URLs are presented to the user (Fig. 25, step 11), 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. This will also be discussed in the harvesting section.

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.


Fig. 32. An Opal request.

### III.5.2 Server Communications

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. Fig. 32 shows an example of an initial Opal request. Appendix 1 gives the full Opal API. To demonstrate usage of the Opal API, several requests to an Opal server, (opal1) are provided and explained below. URL encoding has been removed to improve readability.

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

The request above 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=on&
url=http://www.cs.odu.edu/~tharriso&
new_url=http://www.cs.odu.edu&
c=google&
ia_url=&search=google&page=3
```

This request asks for the Advanced GUI. Both “url” and “new_url” are present and are different, which means the user has modified the URL. Although the next page in the normal sequence should be page 3, (since this request was generated from a page 2), the page=3 request will be over-ridden due to the conflicting URLs. Before new paths can be found (and displayed on page 3), other data points need to be synchronized with the modified URL. To do this, Opal will recalculate a page 2 page with Similar and Source sections for the modified URL, and return this as output to this request. The “c”, “ia_url”, and “search” parameters note options selected on the interface, but are disregarded due to “url” value modification, which requires that Opal synchronize page 2 before proceeding. The following is an advanced request for the “Demo” interface:

```
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
```

This requests to find new paths (page 3) for an unmodified URL (url == new_url ) using the Internet Archive cached version residing at "ia_url". Once the LS is computed, Google (search) will be searched for the result set. Results will be displayed in the Demo (adv) GUI, with TF and IDF calculation output as well (verbose). The Demo GUI is another form of the Advanced interface, therefore both "adv" and "demo" are set to "on".

```
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
```

Voting is currently the only method value in the API. Opal will not change the interface contents and will simply register 1 vote for "url" value URL having a similar URL which is the "match" value. One may recall that a user votes by clicking on a hyperlink that they would like to explore. This click locally calls a JavaScript pop-up window, in addition to casting the vote via CGI. For completeness, a sample href containing this combination is shown below.

```
href=javascript:popUp('opal1.pl?method=vote&
url=http://whiskey.cs.odu.edu/new_position.html&
match=http://www.cs.odu.edu/new_position.html')
```

### III.5.3 Opal/Opal 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 his 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 (Fig. 36). 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 Fig. 33, 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).

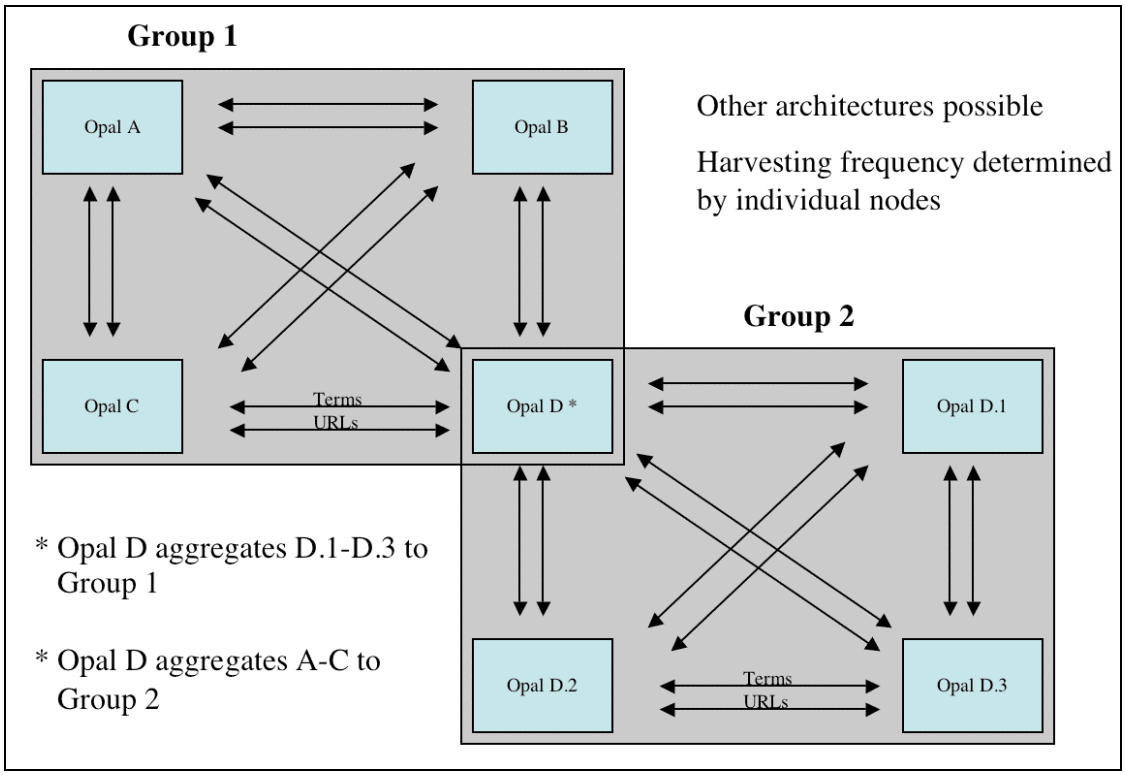



Fig. 33. Opal harvesting.

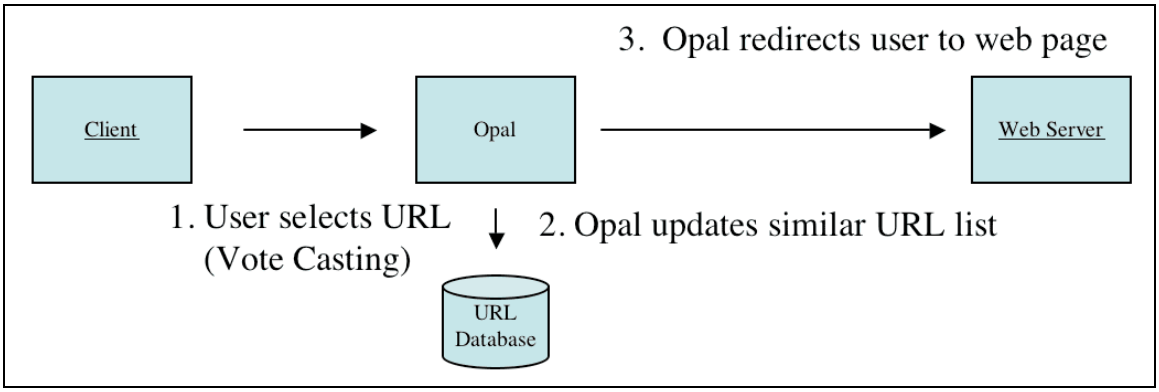



Fig. 34. Voting mechanism.

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, but a URL Db implementation is equally plausible (Fig. 34). The file is later parsed (tallied) via a periodic cron job. Once the tally is concluded and the results combined with the Opal server's earlier knowledge, 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.

## III.6 DATA STRUCTURES

The prototype version of Opal is implemented with several hashes which are tied to the file system using the Unix "dbm" capability. It is important to note that the files that store these hashes are architecture-specific, since they are created according to the byte order of the architecture (Adams, 1995). This limits their portability. A production level Opal server would likely implement these structures utilizing a database, instead of the means used for rapid prototype development.

### III.6.1 Components

Each Opal server consists of seven primary data structure components. These contain the dictionary, similar URLs, metadata, LSs, votes, and an XML Schema for terms as well and another for URLs.

The dictionary is where term data is stored and consists primary mappings from terms to their IDF value. Additional information (see Fig. 36) needed to flush out the term schema are also stored here. In the prototype it is implemented as a hash tied to the file system which is used to serve client requests, but larger scaled implementations

would likely choose to store terms in a database. To illustrate this, the prototype takes the form of (key → value, tab delimited) :

term → server agent datestamp IDF DF corpus_size

A dictionary printout helps to illustrate this (Fig. 35). Some of the data has been rearranged (term count, server, term, agent, datestamp, IDF, DF, corpus_size) by the print script to make the document more readable. URLs which have been voted as similar to the 404 URL are processed during the vote tally and registered as similar URLs. There may be several similar URLs for a 404 URL. Opal servers rank similar URLs according to their votes and may choose to display a fixed number of results. In

Sample Printout of Dictionary:

```
cash.cs.odu.edu:/home/tharriso/public_html/thesis/code % perl printDict.pl
1. Invivo1 757-683-4900 google;2005-06-28-1:28:17;17.516631940177;199;8058044651
2. Invivo1 challenging  google;2005-06-28-1:28:17;5.30554675282329;40000000;8058044651
3. Invivo1 instructor   google;2005-06-28-1:28:17;5.46220056286867;34200000;8058044651
4. Invivo1 virginia     google;2005-06-28-1:28:17;4.24083601583087;116000000;8058044651
5. Invivo1 projects     google;2005-06-28-1:28:17;3.16254372965371;341000000;8058044651
6. Invivo1 details      google;2005-06-28-1:28:17;2.98315903253307;408000000;8058044651
7. Invivo1 this         google;2005-06-28-1:28:17;1.24510374227688;2320000000;8058044651
8. Invivo1 one          google;2005-06-28-1:28:17;1.78656633550476;1350000000;8058044651
9. Invivo1 to           google;2005-06-28-1:28:17;0.952048201763951;3110000000;8058044651
10.Invivo1 i            google;2005-06-28-1:28:17;2.14961072772897;939000000;8058044651
```

Fig. 35. Sample printout of dictionary terms.

the prototype this is implemented as a hash tied to the file system and takes the form (key → value, tab delimited).

404 URL → similarURL1 similarURL2 similarURL3 similarURL4 …

A metadata structure contains brief descriptions of the URLs which are returned in engine result sets. This information is scraped and is associated with its URL as a data record. Initially, this data is logged to a file ( url_metafile.txt ) which is later processed during a vote tally. When a tally lists a URL as a match, its metadata is loaded into a similarURL hash (if it is not already present). In the prototype the similar URL hash is tied to the file system and takes the form (key → value) :

similarURL → (scraped HTML site description)

Lexical signatures are mapped to the 404 URLs. In the prototype this is implemented as a hash tied to the file system and takes the form (key → value) :

404 URL → lexical_signature

Each received vote needs to be logged with server, datestamp, 404 URL, and the URL which has been voted as similar. In the prototype votes are appended to a vote log file. In the prototype, each line in the vote log represents a vote. Data is tab delimited.

VOTE server datestamp 404 URL similarURL

XML Schemas are used by Opal to define unique OAI-PMH “metadataFormats”

for export and harvesting of term and URL related datasets. The Opal prototype incorporates these into Static Repositories (Hochstenbach et al., 2003) which are registered at the Los Alamos National Laboratory Experimental OAI Static Repository Gateway at http://libtest.lanl.gov/. Static Repositories have file size limitations of 2MB, and thus will not be suitable for large scale Opal implementations, which may be more suited to a database driven backend, such as found at the NASA Langley Technical Reports Server (Nelson et al. 2003). Schemas were initially generated using XSDinference (Microsoft, 2002) and then hand refined.

The Term Schema (Fig. 36) defines the structure of XML formatted term data sets which are to be harvested. Fig. 37 exemplifies this Schema in an XML formatted term record. The URL Schema (Fig. 38) defines the structure of XML formatted URL data sets which are to be harvested. A similarURL record is shown in Fig. 39.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns:term="http://www.cs.odu.edu/~tharriso/thesis/code/xml/term.txt"
        targetNamespace="http://www.cs.odu.edu/~tharriso/thesis/code/xml/term.txt"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified">
 <xs:annotation>
            <xs:documentation>Term data extracted from the dictionary</xs:documentation>
 </xs:annotation>

 <xs:element name="term" >
  <xs:complexType>
   <xs:sequence>
    <xs:element name="idf" maxOccurs="unbounded">
     <xs:complexType>
      <xs:attribute name="value" type="xs:decimal" use="required" />
      <xs:attribute name="source" type="xs:string" use="required" />
      <xs:attribute name="agent" type="xs:string" use="required" />
      <xs:attribute name="datestamp" type="xs:date" use="required" />
      <xs:attribute name="df" type="xs:unsignedInt" use="required" />
      <xs:attribute name="corpus" type="xs:unsignedLong" use="required" />
     </xs:complexType>
    </xs:element>
   </xs:sequence>
   <xs:attribute name="value" type="xs:string" use="required"/>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

Fig. 36. Term schema.

```
<term  value="information">
   <idf  value="1.28466934248307"
         source="Invivo2"
         agent="google"
         datestamp="2005-06-30"
         df="2230000000"
         corpus="8058044651">
   </idf>
</term>
```

Fig. 37. XML record for the term “information”.

```
<?xml version="1.0"?>
<xs:schema
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   targetNamespace="http://www.cs.odu.edu/~tharriso/thesis/code/xml/url404.txt"
   elementFormDefault="qualified"
   attributeFormDefault="unqualified">
   <xs:element name="url404">
      <xs:annotation>
         <xs:documentation>url404 XML Schema</xs:documentation>
      </xs:annotation>
      <xs:complexType>
         <xs:sequence>
            <xs:element maxOccurs="unbounded" minOccurs="0" name="similarURL">
               <xs:complexType>
                  <xs:attribute name="datestamp" type="xs:date" use="required"/>
                  <xs:attribute name="votes" type="xs:unsignedByte" use="required"/>
                  <xs:attribute name="simURL" type="xs:string" use="required"/>
                  <xs:attribute name="baseURL" type="xs:string" use="required"/>
               </xs:complexType>
            </xs:element>
         </xs:sequence>
         <xs:attribute name="url" type="xs:string" use="required"/>
      </xs:complexType>
   </xs:element>
</xs:schema>
```

Fig. 38. URL schema.

```
<url:similarURL
   datestamp="2005-06-30"
   votes="4"
   baseURL="http://www.cs.odu.edu/new_position.html"
   <![CDATA[ <p class=q><a href=http://www.cs.odu.edu/new_position.html><b>CS</b>... ]]>
</url:similarURL>
```

Fig. 39. similarURL XML record.

# CHAPTER IV

# DISCUSSION

## IV.1 LEXICAL SIGNATURES

One of the key pieces of Opal is its ability to craft LSs that can find alternate copies of web pages that have gone 404 at a given URL. 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 3 shows LSs created by Opal from live sites (to see if the subsequent search using them would retrieve that site, which they did).

Table 2
Lexical signatures created by Opal

| 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.cs.odu.edu/~jbollen | cs695+dcadl02+jbollen@cs.odu.edu+http://www.cs.odu.edu/~jbollen+bollen |
| http://www.cs.odu.edu/~tharriso | terry+harrison+thesis+jcdl+awarded |
| 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 |

The first few were what the author 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" or words that occur frequently, (but do not really describe a page), 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 highly effective at locating the desired page.

## IV.2 CHALLENGING CONTENT

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 resolve them and avoid this hazard. News sites and other similar pages that change frequently make LS creation challenging for finding

the same site at a different location, but it is possible to find alternate locations for a site which has been updated with more recent content if the older page still exists in another path. Such a case might be where each page is stored with a unique URL, such as joke_july18_2005.html and a more generic main page www.foo.com/joke_of_the_day.html is simply redirected to the previous page.

## IV.3 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".

### IV.3.1 Operational Costs

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.

**Cache location cost:**

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

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 communications.

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:

**New path cost:**

$$Cost_{Paths} = R_c + |T| + R_l$$

For the prototype, the $Cost_{Paths} = 1 + |T| + 1$ or $|T| + 2$ network communications.

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 $(|T| + 6)$ communications. 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 communications 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 the

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.

### IV.3.2 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 (Google, 2005) 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 (TREC, 2002) consisting of 1.6 million unique words (Amati et al., 2001). The Oxford English Dictionary contains about 250000 words or about 750000 if we consider distinct senses, and does not cover technical vocabularies or jargon (Oxford, 2005). 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 what is referred to as either a Zipf or power law distribution (Shirky, 2003), we might hypothesize that the rate at which Opal will run across terms it does not know diminishes 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, each term occurs with a probability (P) that follows the power law distribution $P = Cx^{-n}$ (Adamic, 2002). For reasons of simplicity, we chose C=1 and n=-1. This means a few terms (e.g. "a", "an", and "the") will have a high probability of being selected, while other, more rarified terms will have a very low probability of selection.

We ran tests with the following parameters:

- 1 Million terms assumed as number of terms in lexicon
- 30000 Document collection
- Even and Zipf distributions with 100 and 200 term documents
- Results are the averages of 100 iterations of each test

Fig. 40 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.

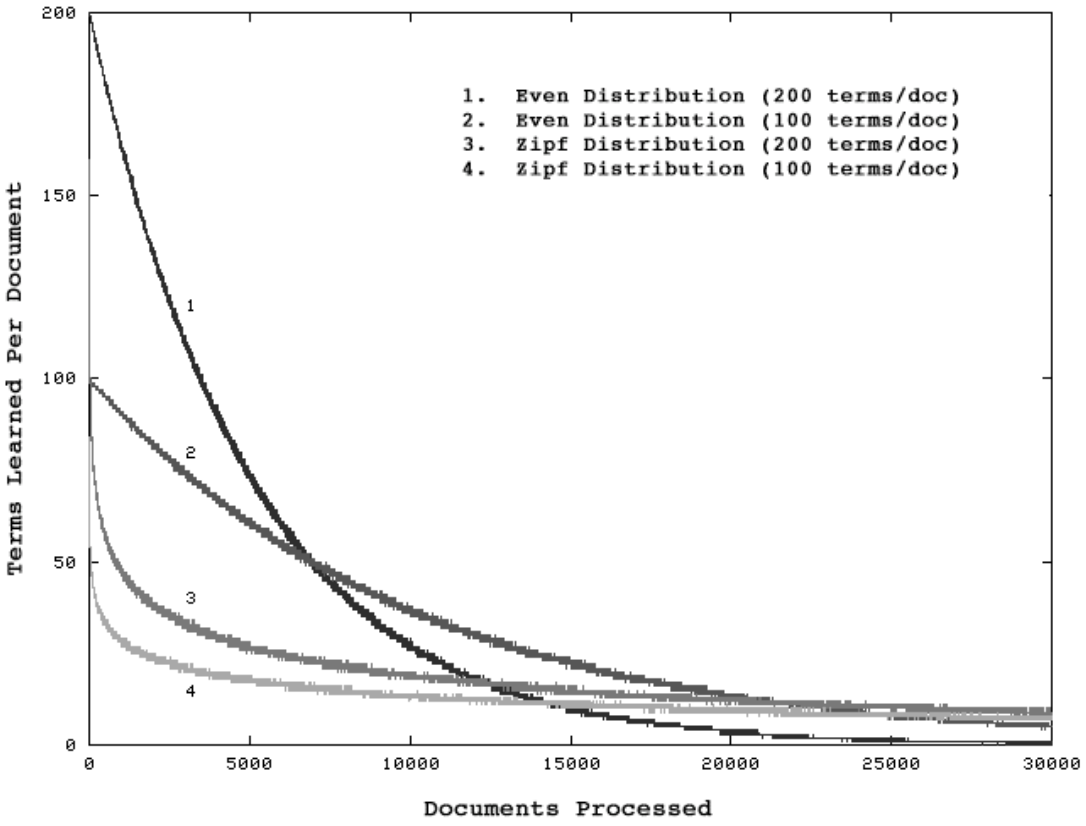



Fig. 40. Terms learned per document.

Fig. 41 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. If a document is large, but is filled with articles, it's number of unique terms is smaller still.

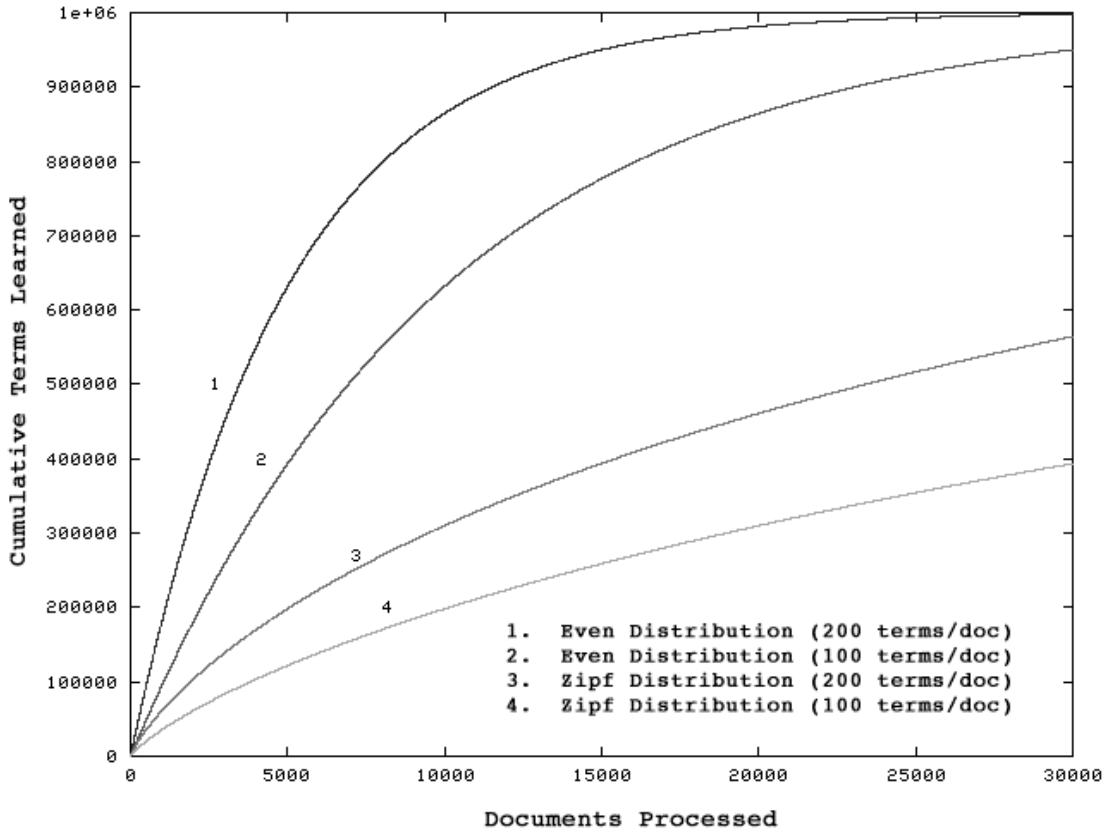



Fig. 41. Cumulative terms learned.

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, we see that even after 30000 documents have been parsed, we are still learning new terms (Fig. 41) and the rate of about 10 per document, with small changes anticipated in this rate, even with the

ingestion of greater numbers of documents.

### IV.3.3 Load Estimation

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 (Fig. 42). Of 4.5 million http requests, there were approximately 500000 unique URL requests, and 75000 total 404 unique URLs, as detailed in Table 4. 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.

Table 3
ODU CS log totals for May 2005

| | |
|---|---|
| Total http requests | 4530218 |
| Total unique URL requests | 557110 |
| Total 404 errors | 457917 |
| Total 404 unique URLs | 74877 |

A quick look at the logs show a power-law type distribution of the requests for the 404 URLs, where 25% of the totals were for the top entry, the “favicon.ico”. The “favicon.ico” is the small icon that appears in the browser URL bar, (e.g. the little Y! that appears when you go to www.yahoo.com). Fifth on the list is

"/WWW/documents/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 4).

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.

```
104293 /WWW/documents/favicon.ico
 36686 /WWW/documents/files/shadow-b.gif
 36571 /WWW/documents/files/shadow-r.gif
 10600 /home/edenecke/public_html/gallery/swf/games/crashtest.swf
  7881 /WWW/documents/robots.txt
  7103 /home/rtrifono/public_html/counter/files/update.pl
  6342 /home/rtrifono/public_html/counter_intladm/files/update.pl
  6307 /WWW/documents/final/files/shadow-b.gif
  6185 /WWW/documents/final/files/shadow-r.gif
  5811 /WWW/documents/images/back.gif
  2652 /home/consult/public_html/rootmail.wav
  2632 /WWW/documents/~tzu/life.php
  1414 /home/cpi/public_html/cpi95-96/chalk.jpg
  1398 /home/cs350/public_html/sp05/white
  1255 /home/wild/public_html/cs477/spring04/images/rainbow.gif
  1158 /home/edenecke/public_html/gallery/swf/cartoons/usar.swf
  1114 /home/cmorris/public_html/img/background/nbg.gif
  1065 /home/edenecke/public_html/gallery/swf/games/gravityball.swf
   983 /home/vsripada/public_html/mile
   937 /home/cs101/public_html/summer05/CS
   891 /home/advisor/public_html/program/images/redbar.gif
   797 /home/cs333/public_html/summer05/assignments/background.gif
   776 /WWW/documents/_vti_inf.html
   769 /WWW/documents/_vti_bin/shtml.exe/_vti_rpc
   717 /home/cs333/public_html/summer05/assignments/program.css
   700 /WWW/documents/config/login
   684 /home/edenecke/public_html/gallery/swf/cartoons/morto.swf
   642 /home/cmorris/public_html/new/html/faq.js
   640 /home/cmorris/public_html/new/images/odu/gfx-btn-go-dblue.gif
   638 /home/cmorris/public_html/new/images/odu/dot-1x1-mblue.gif
   591 /home/jwogan/public_html/untitled.bmp
   562 /home/jlazerni/public_html/babe1.jpg
   559 /home/jlazerni/public_html/babe2.jpg
   551 /home/jlazerni/public_html/babe3.jpg
   544 /home/edenecke/public_html/gallery/swf/games/escopeta.swf
   543 /home/vcrinkla/public_html/advise/web/files/shadow-b.gif
   543 /home/vcrinkla/public_html/advise/web/files/hmenu_bg_dept1.gif
   543 /home/jlazerni/public_html/babe4.jpg
   540 /home/vcrinkla/public_html/advise/web/files/lmenu_bg_162.gif
   538 /home/vcrinkla/public_html/advise/web/files/shadow-r.gif
   532 /home/dgarcia/public_html/tuto/jscripts.js
   501 /WWW/documents/final/files/lmenu_bg_162.gif
```

Fig. 42. ODU CS department 404 log file for May, 2005.

Table 4
Opal request load

| | |
|---|---|
| 457917 | Total 404 errors |
| 74877 | Roughly 15% of the unique URL requests are for URLs which have gone 404 |
| 56157 | 75% of the logged 404 errors not from machine made requests or crawlers |
| 5615 | 10% of these are cached or have already had an LS calculated by Opal |
| 187 | Requests per day that may require term lookup (dividing above number into 30 days) |

### IV.3.4 Scalability

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”.

## IV.4 SECURITY CONSIDERATIONS

Security is always a concern for systems offering information services on the Web. Intentional attacks such as intruders posing as Opal or contemplating 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.

# CHAPTER V

# FUTURE WORK

The Opal prototype demonstrated the potential of the Opal framework, but a great deal of exciting work remains. Further large 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 (Pandey et al., 2005).

## V.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. Source code and guidelines are readily available from the OAI for implementing OAI-PMH repositories (Lagoze et al., 2002). 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 (Habing et al., 2004). If $Opal_1$ harvests $Opal_{A...C}$ while $Opal_2$ harvests $Opal_{C...G}$ , then it might make more sense for $Opal_3$ to harvest $Opal_1$ and $Opal_2$ (even with the redundancy of the $Opal_C$ data) rather than harvest $Opal_1$, $Opal_D$, $Opal_E$, $Opal_F$, and $Opal_G$ separately. By using "Sets" $Opal_3$ could harvest from $Opal_1$ and then $Opal_2$, but request only sets D,E,F,G from $Opal_2$, 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. We have determined that some pages, notably media-rich sites like those built with Flash and sites 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.

## V.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. (2004) 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)?

Park et al. (2004) expanded on the LS work of Phelps & Wilensky, but did not explicitly investigate how LSs 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. Another interesting example is show in Fig. 43. The web page for the 2000 ACM Digital Library Conference, www.dl00.org, has not disappeared. But the conference did not renew the domain registration, and in late 2001 the domain was "hijacked". The URL www.dl00.org has not (yet) become unavailable, but when it does, the LS generated by consulting the Wayback Machine is likely to be surprising. We are interested in applying trend analysis to detect significant shifts in the page aboutness, similar to the techniques applied in the Walden's PathProject (Dalal et al., 2004).

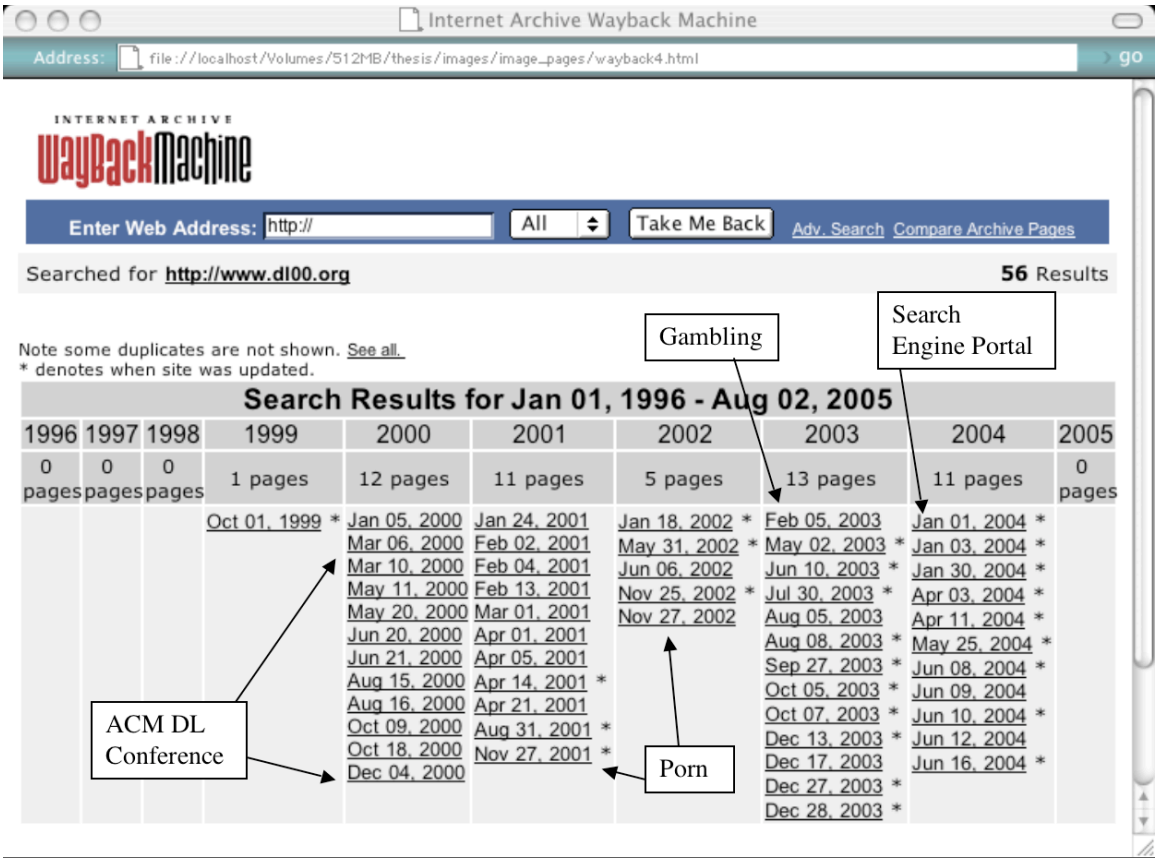



Fig. 43. Different cached versions from the Wayback Machine.

It is worthy to reiterate that lexical signatures only work with textual content. It would be interesting to have a similar concept available for non-textual material, such as video. Using checksums, hashing, or fingerprinting techniques could work, but would only be suited for locating exact matches. Searching for "similar" videos and other such media remains an area of open research.

## V.3 IDF

We also plan to investigate additional IDF evaluation techniques, such as estimation

based on limited crawls of hyperlinked neighboring pages (Sugiyama et al., 2003). Scraping Google for IDF values might be 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? Is the name of Libya's leader "Khadafi" or "Qadafi"?

Additionally, different languages may yield different index terms from the same extraction process. Languages are often transliterated syntactically, which does not completely translate the meaning. Special punctuations which may be used to give otherwise similar terms different meanings may be dropped. When translation dictionaries are utilized, errors may creep in where words have several meanings. They may also lack technical jargon in their lexicon. Finally, different languages have different concepts of what a term is. The German language, for example, is rich with concatenated words that have no spaces between them; making it even more complex to perform indexing in a word based sense (Haddouti, 1998).

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.

## V.4 OPAL AS AN OPENURL RESOLVER

OpenURL is about providing localized services on metadata. The 404 URL we are passing in could be described as metadata, and exploration of the various caches as a type of service provided by an OpenURL resolver (Van De Sompel & Beit-Arie, 2001). In such a manner, Opal may be able to be implemented as an OpenURL resolver.

## V.5 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 (Christensen et al., 2001). Since WSDL is encoded in XML, we can use OAI-PMH to harvest WSDL records from each server. We can attach XML signatures (Eastlake, Reagle, & Solo, 2002) 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.

# CHAPTER VI

# CONCLUSIONS

Opal has demonstrated in flexible framework for locating alternative pages that are similar to URLs that have gone 404. Opal takes advantage of the Web Infrastructure as a mechanism to locate caches, learn term IDF weights for the web, and to search using lexical signatures that Opal derives. Opal provides a service that encompasses many individual search services and leverages their collective knowledge. Opal can share and learn data utilizing OAI-PMH. Opal does not harvest data sets for redistribution of the data to users, like most harvesters, but rather Opal gathers this data to increase its own knowledge, which it applies to its task of LS building and site searching. Using the OAI-PMH to facilitate machine to machine learning extends the utilization of OAI-PMH in a manner that has not been done before.

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 into Opal is warranted.

Opal brings together services from the WI (search engines), lexical signatures, along with OAI-PMH as a communications vehicle to promote invivo preservation. By locating data that was designated as 404 by the client's web server, we retie the string connecting the client and the end data… thus preserving the connection.

# REFERENCES


Adamic, L. A. (n.d.). Zipf, power-laws, and Pareto - a ranking tutorial. Retrieved Jul. 10, 2005, from: http://www.hpl.hp.com/research/idl/papers/ranking/ranking.html.

Adams, D. (1995). Unix man pages: dbm (3UCB). Retrieved on July 05, 2005, from: http://www.cs.biu.ac.il/cgi-bin/man?dbm+3UCB.

Amati, G., Carpineto, C., & Romano G. (2001). FUB at TREC-10 web track: a probabilistic framework for topic relevance term weighting. In *Proceedings of TREC 2001*.

Berners-Lee, T. (1998). Cool URIs don't change. Retrieved on July 10, 2005, from: http://www.w3.org/Provider/Style/URI.html.

Bollen, J. (2003). Topics in information retrieval. Retrieved on April 21, 2005, from: http://www.cs.odu.edu/~jbollen/spring03_IR/slides.html.

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

Brin, S., Davis, J., & Garcia-Molina, H. (1995). Copy detection mechanisms for digital documents. In *Proceedings of the ACM SIGMOD annual conference* (pp. 398-409).

Chan, S., McCown, F., Nelson, M., & Bollen, J. (submitted for publication). The availability and persistence of web references in D-Lib magazine. *5th international web archiving workshop and digital preservation*. Available at http://www.cs.odu.edu/~fmccown/dlib-persistence-fm.pdf.

Christensen, E., Curbera, F., Meridith, G., & Weerawarana, S. (2001). Web services description language (WSDL) 1.1. Retrieved on July 10, 2005, from: http://www.w3.org/TR/wsdl.

Cho, J., & Garcia-Molina, H. (2003). Effective page refresh policies for Web crawlers. *ACM Transactions on Database Systems*, 28(4), 390-426.

Cho, J., Shivakumar, N., & Garcia-Molina, H. (2000). Finding replicated Web collections. In *Proceedings of SIGMOD conference 2000* (pp. 355-366).

CrossRef, (2004). CrossRef launches pilot program of CrossRef search, Powered By Google, Retrieved on July 10, 2005, from: http://www.crossref.org/01company/pr/press20040428.html.

Dalal, Z., Dash S., Dave P., & Francisco-Revilla, L. (2004). Managing distributed collections: evaluating web page changes, movement, and replacement. In *Proceedings of the 4th ACM/IEEE joint conference on digital libraries* (pp. 160-168).

Eastlake, D., Reagle, J., & Solo, D. (2002). XML-signature syntax and processing. Internet RFC 3275. Retrieved on July 10, 2005, from: http://www.ietf.org/rfc/rfc3275.txt.

Google, (2005). Google web APIs. Retrieved on July 10, 2005, from: http://www.google.com/apis/api_faq.html#gen12.

Gulli, A., & Signorini, A. (2005). The indexable web is more than 11.5 billion pages. In *Proceedings of WWW '05* (pp. 902-903).

Habing, T., Cole, T., & Mischo, W. (2004). Developing a technical registry of OAI data providers. In *Proceedings of the 8th European conference on digital libraries* (pp. 400-410).

Haddouti, H. (1998). Multilinguality issues in digital libraries. In *Proceedings of the EuroMed Net'98 conference*.

Hochstenbach, P., Jerez H., & Van de Sompel, H. (2003). The OAI-PMH static repository and static repository gateway. In *Proceedings of the 3rd ACM/IEEE-CS joint conference on digital libraries* (pp. 210-217).

Internet Archive (2005). About the Wayback machine. Retrieved on July 10, 2005, from: http://www.archive.org/web/web.php.

Kahle, B. (1997). Preserving the internet. *Scientific American*, 276(3), 82-83.

Koehler, W. (1999). An analysis of web page and web site constancy and permanence. *Journal of the American Society for Information Science*, 50(2), 162-180.

Koehler, W. (2004). A longitudinal study of Web pages continued: a consideration of document persistence. *Information Research*, 9(2).

Lagoze, C., Van De Sompel, H., Nelson, M. L., & Warner, S. (2002). Implementation guidelines for the Open Archives Initiative for Metadata Harvesting: specification and XML Schema for the OAI identifier format. Retrieved of July 10, 2005, from: http://www.openarchives.org/OAI/2.0/guidelines-oai-identifier.htm.

Lagoze, C., & Van De Sompel, H. (2001). The Open Archives Initiative: building a low-barrier interoperability framework. In *Proceedings of the 1st ACM/IEEE joint conference on digital libraries* (pp. 54-62).

Lawrence, S., Coetzee, F., Glover, E., Pennock, D., Flake, G., Nielsen, F., Krovetz, R., Kruger, A., & Giles, C. L. (2001). Persistence of web references in scientific research. *IEEE Computer*, 34(2), 26-31.

Maniatis, P., Roussopoulos, M., Giuli, T. J., Rosenthal, D. S. H., Baker, M., & Muliadi, Y. (2003) Preserving peer replicas by rate-limited sampled voting, In *Proceedings of the 19th ACM symposium on operating systems principles* (pp. 44-59).

Mauildin, M., & Schwartz, M. (1996). Spidering BOF report. Retrieved on July 18, 2005, from: http://www.w3.org/Search/9605-Indexing-Workshop/ReportOutcomes/Spidering.txt.

Microsoft (2002). Microsoft XSD Inference 1.0. Retrieved on July 10, 2005, from: http://apps.gotdotnet.com/xmltools/xsdinference/.

Mimech (2003). How search engines and crawlers work. Retrieved on May. 1, 2003, from: http://mimech.com/search%2Dengines/.

Nelson, M. L., Harrison T. L., & Rocker J. (2003). OAI and NASA's scientific and technical information. *Library Hi Tech*, 21(2), 140-150.

Nelson, M. L. (2001). Better interoperability through the Open Archives Initiative. *New Review of Information Networking,* 7, 133-145.

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

Oxford University Press (2005). Frequently asked questions. Retrieved on July 10, 2005, from: http://www.askoxford.com/asktheexperts/faq/aboutenglish/numberwords.

Pandey, S., Roy, S., Olston, C., Cho, J., & Chakrabarti, S. (2005). Shuffling a stacked deck: the case for partially randomized ranking of search engine results. In *Proceedings of the 31st international conference on very large databases*.

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

Phelps, T. A., & Wilensky, R. (2000). Robust hyperlinks cost just five words each. UCB Computer *Science Technical Report*, UCB//CSD-00-1091.

Reich, V., & Rosenthal, D. S. H. (2001). LOCKSS: A permanent Web publishing and access system, *D-Lib Magazine*, 7(6).

RLG (1996). Preserving digital information: Report of the task force on archiving of digital information. Retrieved on July 10, 2005, from: http://www.rlg.org/ArchTF/.

Rothenberg, J. (1999). Avoiding technological quicksand: finding a viable technical foundation for digital preservation, CLIR Report #77 1999. Retrieved on July 10, 2005 from: http://www.clir.org/pubs/abstract/pub77.html.

Salton, G., & Buckley, C. (1988) Term-weighing approaches in automatic text retrieval. *Information Processing and Management,* 24(5), 513-523.

Shirky, C. (2003). Power laws, weblogs, and inequality. Retrieved on July 10, 2005, from: http://www.shirky.com/writings/powerlaw_weblog.html.

Shivakumar, N., & Garcia-Molina, H. (1996). Building a scalable and accurate copy Detection Mechanism. In *Proceedings of the 1st ACM digital library conference* (pp. 160-168).

Shivakumar, N., & Garcia-Molina, H. (1998). Finding near-replicas of documents and servers on the Web. In *Proceedings of WebDB 1998* (pp. 204-212).

Spinellis, D. (2003). The decay and failures of web references. *Communications of the ACM*, 46(1), 71-77.

Sugiyama, K., Hatano, K., Yoshikawa, M., & Uemura, S. (2003). Refinement of TFIDF schemes for Web pages using their hyperlinked neighboring pages. In *Proceedings of hypertext 2003* (pp. 198-207).

TREC, (2002). TREC web corpus: wt10g. Retrieved July 10, 2005, from: http://es.csiro.au/TRECWeb/wt10g.html.

Twist, J. (2004, Sept 24). Web tool may banish broken links. BBC News. Retrieved February 24, 2005, from: http://news.bbc.co.uk/1/hi/technology/3666660.stm.

Van De Sompel, H., & Beit-Arie, O. (2001). Generalizing the OpenURL framework beyond references to scholarly works. *D-Lib Magazine*, 7(7/8).

Van De Sompel, H., Young, J. A., & Hickey, T. B. (2003). Using the OAI-PMH … differently. *D-Lib Magazine,* 9(7/8).

Van De Sompel, H., & Lagoze, C. (2001). The Open Archives protocol for metadata harvesting. Retrieved on June 10, 2005, from: http://www.openarchives.org/OAI/openarchivesprotocol.html.

Van De Sompel, H., & Lagoze, C. (2002). Notes from the interoperability front: A progress report on the Open Archives Initiative. In *Proceedings of the 6th European conference on digital libraries* (pp. 144-157).

Ward, M. (2000). Web links that stick. BBC news. Retrieved February 24, 2005, from: http://news.bbc.co.uk/1/hi/sci/tech/790685.stm.

Yahoo, (2004). U-M expands access to hidden electronic resources with OAIster. Retrieved July 10, 2005 from: http://www.umich.edu/news/?Releases/2004/Mar04/r031004.

Yotov, K. (2001). Web crawlers. Retrieved February 22, 2001, from: http://www.cs.cornell.edu/Courses/cs430/2001SP/slides/lecture10.ppt.

# APPENDIX

## OPAL API

The following is a listing of the Opal query names, values (may be implementation specific) and a brief description of the purpose of each:

Table 5
Opal API

| Name | Value | Description |
|---|---|---|
| method | vote | indicates that the data passed pertains to a vote |
| adv | on \| NULL | to engage/disengage the advanced user interface |
| demo | on \| NULL | to engage/disengage the demo user interface |
| verbose | on \| NULL | to display term frequency and IDF calculation details |
| search | google | name of search engine to search with LS |
| c | google \| yahoo \| ia | name of source providing cache |
| version | Cached \| Live | Live – use URL provided (and not cached version)<br>Cached – use a cached version of the URL |
| new_url | URL | URL submitted via the text box \| initial redirect 404 URL |
| url | URL | the 404 URL that initiated the previous page |
| ia_url | URL | user provided URL to desired Internet Archive cache |
| match | URL | URL which user has voted as similar/matching |
| page | 2 \| 3 | indicates which GUI screen to proceed to |

# VITA

Terry L. Harrison

Department of Computer Science

Old Dominion University

Norfolk, VA 23529

## Education

- MS, 2005, Computer Science, Old Dominion University, Norfolk, VA
- BS, 1992, Mass Communications, James Madison University, Harrisonburg, VA

## Employment

- Information Systems Support, Inc., Research Analyst, January 2004 – present.
- Old Dominion University, Research Assistant, January 2002 - present
- XMLSolutions, Technology Instructor, April 2001 – April 2002.
- Anheuser Busch, Audio-Video Technician, 1998 – 2002.
- Optical Imagineering, Owner, 1996 – 1998.
- WPEN-TV, Producer-Engineer, 1993 – 1996.

## Professional Affiliation

- IEEE Computer Society
- Phi Kappa Phi Honor Society