Web Archiving Incentive Program: Final Report

Warcbase: An Open-Source Platform for Managing Web Archives

Jimmy Lin
University of Maryland and University of Waterloo
jimmylin@umd.edu, jimmylin@uwaterloo.ca

November 10, 2015

1 Background

The goal of the Warcbase project is to build an open-source platform for storing, managing, and analyzing web archives using modern "big data" infrastructure—specifically, the Hadoop Distributed File System (HDFS) for storage, HBase for providing random access to archived content, and Hadoop for data analytics. The proposal had four goals:

1. Development of the core HBase data model.
2. Construction of an application for temporal browsing.
3. Implementation of basic analytics capabilities using Pig.
4. Prototyping full-text search capabilities.

This final report is organized in terms of the four project goals. The first three goals were part of the original award (April 1, 2014 to December 31, 2014); the final goal was part of a continuation award (January 1, 2015 to November 15, 2015). This document is written in a cumulative fashion, describing accomplishments throughout the entire project (including progress already reported in a interim report dated November 14, 2014).

The Warcbase homepage is located at `http://warcbase.org/`, which redirects to a GitHub repository that holds the Warcbase codebase and all associated documentation. All material is released under the Apache License, per terms of the original award. Development of this project has taken place (and will continue to take place beyond this award) in a completely open environment: anyone can inspect our code at any point in time, follow progress in our issues tracker,[1] open new tickets requesting features or bug fixes, or contribute patches if they feel inclined.

At the beginning of the award, the PI was a faculty member at the University of Maryland, but as of August 2015, the PI moved to the University of Waterloo. During the course of this project, the PI developed a collaboration with Ian Milligan, Assistant Professor in History at the University of Waterloo. Some of the accomplishments reported here are the direct result of this fruitful collaboration.

2 Working Data and Hadoop Cluster Resources

Warcbase was developed primarily with the following datasets:

- We have access to a crawl of the 108th U.S. Congress gathered by the Library of Congress, made available to the University of Maryland via a resource-sharing agreement. It can be characterized as a "narrow but deep" crawl of the websites of members of the House of Representatives and the Senate from January 2003 to January 2005 at monthly intervals. The collection totals 1.15 TB of gzipped ARC files, containing approximately 29 million captures of 7.8 million unique URLs; of those, 23.8 million captures are HTML pages.

[1] `https://github.com/lintool/warcbase/issues`

- We have also worked with the Canadian Political Parties and Political Interest Groups collection, gathered by the University of Toronto Library using the Internet Archive's Archive-It platform.[2] The collection contains 14.5 million documents from crawls performed at roughly quarterly intervals between October 2005 and March 2015. Content from around fifty organizations were collected: all of the major Canadian political parties (the Conservative Party, the Liberal Party, the New Democratic Party, the Green Party, and the Bloc Quebecois), as well as minor parties and political organizations such as the Assembly of First Nations, the Canadian Association for Free Expression, Fair Vote Canada, and beyond. The entire collection totals 380 GB in size compressed, comprised of both ARC and WARC files.

- For smaller-scale testing, we have access to some of Columbia University's web archive collections, including the Human Rights Web Archive. These data represent a mixture of ARC and WARC files.

Development has been conducted on a Hadoop (YARN) and HBase cluster running Cloudera's Distribution of Hadoop (CDH) at the University of Maryland.

3 Project Accomplishments

3.1 Development of the Core HBase Data Model

Data Modeling and Schema Design. We have successfully designed and implemented the core Warcbase data model.

First, a quick overview of HBase schema design: the system is best described as a sparse, persistent, multiple-dimensional sorted map. An HBase table maintains a mapping from a 4-tuple to arbitrary values as follows:

$$(\text{row key, column family, column qualifier, timestamp}) \rightarrow \text{value}$$

Conceptually, values (arbitrary bytes) are identified by rows and columns. A row is identified by its row key. Rows are lexicographically sorted by row key. Each column is decomposed into "family" and "qualifier" (written as "family:qualifier"), where the family provides a mechanism for the developer to specify groupings of qualifiers that should be physically co-located. The timestamp allows multi-versioned values.

After exploring a few alternatives (more below), we ultimately implemented the design described in the original project proposal. Warcbase keeps each archival collection in a separate HBase table. We assume that each resource (e.g., web page, image, PDF, etc.) can be uniquely identified by its URL and a timestamp (the crawl date).

A resource's domain-reversed URL serves as the key, i.e., `www.house.gov/index.html` becomes `gov.house.www/index.html`. This design allows different resources from the same domain to be physically co-located on disk, and thus client requests can benefit from reference locality. The raw content of the resource (HTML, PDF, image, etc.) is stored as the value in the column family "c" with the MIME type as the qualifier (e.g., "text/html"). Each version of the resource (corresponding to a different crawl) has a unique timestamp.

During the development process we had considered a few alternative designs:

- Our data model is colloquially termed a "fat row" design because each row can grow quite large (since it stores all captures of the resource). We had considered an alternative that explicitly stores the timestamp as part of the row key, for example,

[2] `https://archive-it.org/collections/227`

```
gov.house.www/index.html+20031224215817,
```

so that different versions of the resource reside in separate HBase rows. This design, not surprisingly, is termed "thin rows" since each row only holds a capture. Although fat row designs are less common in HBase applications, we nevertheless decided to adopt the fat row approach because explicitly storing the timestamp in the row key seemed verbose and wasteful.

- To render a resource properly, we need to know its MIME type, since this is part of the server response in the HTTP protocol. Thus, we need to store the MIME type of the resource somewhere in Warcbase. The first option is to store it as the value of a separate column qualifier, in which case we would need two separate queries to properly render a resource (the first query to get the MIME type, the second query to fetch the actual content). The second option is to store the MIME type as the column qualifier itself. The advantage here is that we don't need a separate query to determine the MIME type, but instead we need to execute a short range scan (of column qualifiers associated with that row) to fetch the content (because we might not know *a priori* the MIME type[3]). We opted for the second design since range scans are well supported in HBase.

Finally, we experimented with different compression schemes for the physical HBase data layout and settled on Snappy compression as a good compromise between compression ratio and speed.

Ingestion of Archived Data. We have implemented a program called `IngestFiles`[4] that ingests both ARC and WARC files into Warcbase. We have successfully ingested all of the 108th Congress collection (1.15 TB) as well as the Canadian Political Parties and Political Interest Groups collection (380 GB) into our running instance of Warcbase. Details of how to invoke the ingestion program can be found in the Warcbase wiki.

Hadoop Bindings. To enable analytics using MapReduce (and also Pig), it is necessary to implement bindings (or "adaptors") for reading web archive data. We have built such bindings for ARC and WARC data stored directly on the Hadoop Distributed File System (HDFS) as well as data that have been ingested into Warcbase. These bindings take advantage of Hadoop's `InputFormat` specification, which defines a generic interface for reading key–value pairs from a data stream, which could come from HBase or HDFS directly.

URL Mapping Dictionary. While resources in web archives can be uniquely identified by URLs, they are not ideal for data processing: URLs can be quite long and thus require substantial memory to store. It would be desirable to represent each URL with a unique integer, which can be far more compactly stored. We have implemented a program to build exactly such a mapping using the open-source Lucene search engine's finite state transducer (FST) package.[5]

Details of how to build this URL mapping dictionary can be found in the Warcbase wiki.

3.2 Construction of an Application for Temporal Browsing

We have successfully integrated Warcbase with the OpenWayback system[6] to provide temporal browsing capabilities. Through this integration, any installation of the OpenWayback system can transparently browse web archives stored in Warcbase. This is made possible by two key abstractions that the OpenWayback provides:

[3]File extensions are often unreliable in this regard.

[4]`https://github.com/lintool/warcbase/blob/master/src/main/java/org/warcbase/ingest/`

[5]`http://lucene.apache.org/`

[6]`https://github.com/iipc/openwayback`

- **Resource Store**, which is responsible for retrieving archived content, abstracting the medium and format used to store the web content.
- **Resource Index**, which is responsible for satisfying queries against the documents located in the resource store, e.g., requests for a particular timestamped version of a URL.

In a standard installation, the resource store provides access to WARC and ARC data stored on the file system (either local disk or network-attached storage). Alternatively, a resource store can access individual resources via a remote service. Similarly, OpenWayback provides alternative implementations of the resource index: a local BerkeleyDB index, a local CDX index, and a remote resource index. Small-scale installations typically use the BerkeleyDB index.

Integration of Warcbase and OpenWayback is accomplished by providing custom implementations of the resource store and the resource index. In Warcbase, the implementations are contained in the `org.warcbase.wayback` package.[7] Specifically, the `WarcbaseResourceStore` and `WarcbaseResourceIndex` classes connect with a REST API provided by a class called the `WarcBrowser`,[8] which allows the OpenWayback to query for resources (e.g., lookup a particular version of a URL) and to fetch relevant content.

For example, if a user requested the URL `http://www.house.gov/` from the OpenWayback interface, the resource index would issue the following REST API call to the `WarcBrowser` (assuming the REST API is running on port 8080 of the machine `nest.umiacs.umd.edu`):

```
http://nest.umiacs.umd.edu:8080/c108th/*/http://www.house.gov/
```

The response would be the following:

```
20031224215817 text/html /c108th/20031224215817/http://www.house.gov/
20040124034300 text/html /c108th/20040124034300/http://www.house.gov/
20040225033035 text/html /c108th/20040225033035/http://www.house.gov/
20040324010018 text/html /c108th/20040324010018/http://www.house.gov/
20040424001734 text/html /c108th/20040424001734/http://www.house.gov/
...  (16 lines in total)
```

which enumerates all the versions of the resource that are available (16 total). The first column shows the timestamp, in the standard 14-digit format (e.g., 2003/12/24 21:58:17); the second column specifies the MIME type of the resource; the third column specifies the resource path for fetching the resource content.

The OpenWayback instance would render these results in the familiar interface (Figure 1, left). If the user selects, for example, the first version, OpenWayback would request the relevant resource via the resource store. The Warcbase implementation of the resource store would then issue the following REST API call:

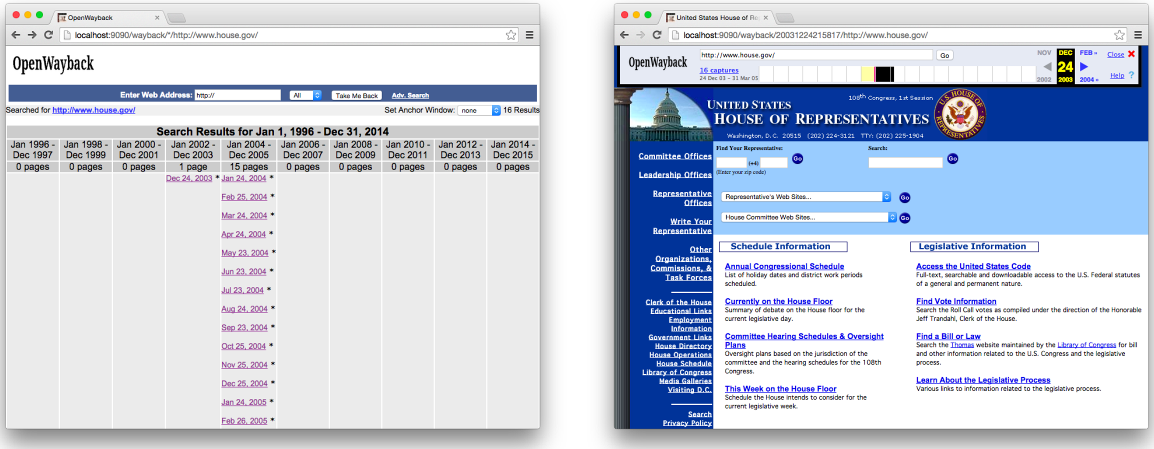
```
http://nest.umiacs.umd.edu:8080/c108th/20031224215817/http://www.house.gov/
```

which returns the raw HTML content of the particular version of that page (Figure 1, right).

The OpenWayback/Warcbase integration provides a seamless browsing experience that is identical to any other OpenWayback installation. However, the advantage is that, via Warcbase, storage

[7] `https://github.com/lintool/warcbase/tree/master/src/main/java/org/warcbase/wayback`

[8] `https://github.com/lintool/warcbase/tree/master/src/main/java/org/warcbase/browser`



OpenWayback

localhost:9090/wayback/*/http://www.house.gov/

OpenWayback

Enter Web Address: http:// All Take Me Back Adv. Search

Searched for http://www.house.gov/ Set Anchor Window: none 16 Results

Search Results for Jan 1, 1996 - Dec 31, 2014

Jan 1996 - Dec 1997	Jan 1998 - Dec 1999	Jan 2000 - Dec 2001	Jan 2002 - Dec 2003	Jan 2004 - Dec 2005	Jan 2006 - Dec 2007	Jan 2008 - Dec 2009	Jan 2010 - Dec 2011	Jan 2012 - Dec 2013	Jan 2014 - Dec 2015
0 pages	0 pages	0 pages	1 page	15 pages	0 pages	0 pages	0 pages	0 pages	0 pages
			Dec 24, 2003 *	Jan 24, 2004 *					
				Feb 25, 2004 *					
				Mar 24, 2004 *					
				Apr 24, 2004 *					
				May 23, 2004 *					
				Jun 23, 2004 *					
				Jul 23, 2004 *					
				Aug 24, 2004 *					
				Sep 23, 2004 *					
				Oct 25, 2004 *					
				Nov 25, 2004 *					
				Dec 25, 2004 *					
				Jan 24, 2005 *					
				Feb 26, 2005 *					

United States House of Re

localhost:9090/wayback/20031224215817/http://www.house.gov/

OpenWayback http://www.house.gov/ Go

16 captures 24 Dec 03 - 31 Mar 05

NOV 2002 DEC 24 2003 FEB » 2004 » Close Help

108th Congress, 1st Session

UNITED STATES HOUSE OF REPRESENTATIVES

Washington, D.C. 20515 (202) 224-3121 TTY: (202) 225-1904

Committee Offices
Leadership Offices
Representative Offices
Write Your Representative
Other Organizations, Commissions, & Task Forces
Clerk of the House
Educational Links
Employment Information
Government Links
House Directory
House Operations
House Schedule
Library of Congress
Media Galleries
Visiting D.C.
Search
Privacy Policy

Find Your Representative: (+4) Go (Enter your zip code)

Search: Go

Representative's Web Sites... Go

House Committee Web Sites... Go

Schedule Information

Annual Congressional Schedule
List of holiday dates and district work periods scheduled.

Currently on the House Floor
Summary of debate on the House floor for the current legislative day.

Committee Hearing Schedules & Oversight Plans
Oversight plans based on the jurisdiction of the committee and the hearing schedules for the 108th Congress.

This Week on the House Floor
Schedule the House intends to consider for the current legislative week.

Legislative Information

Access the United States Code
Full-text, searchable and downloadable access to the U.S. Federal statutes of a general and permanent nature.

Find Vote Information
Search the Roll Call votes as compiled under the direction of the Honorable Jeff Trandahl, Clerk of the House.

Find a Bill or Law
Search the Thomas website maintained by the Library of Congress for bill and other information related to the U.S. Congress and the legislative process.

Learn About the Legislative Process
Various links to information related to the legislative process.

Figure 1: Screenshots illustrating OpenWayback/Warcbase integration, showing captures from the U.S. 108th Congress.

management is offloaded to HBase, which automatically handles allocation of physical storage, management of compression, indexing for random access, replication for fault tolerance, etc. Collections can grow arbitrarily large in size without worrying about the scalability limitations of BerkeleyDB or CDX-based indexing.

Details of how to set up and configure the OpenWayback/Warcbase integration can be found in the Warcbase wiki.

3.3 Implementation of Basic Analytics Capabilities Using Pig

We have successfully developed analytics capabilities in Pig over web archive data (both ARC and WARC). The implementation consists of two main components: language bindings that allow Pig to read ARC and WARC data, and user-defined functions (UDFs) that encapsulate specific operations on web archive data. Examples of UDFs that we have written include those to manipulate URLs, extract links, and extract textual content from HTML pages (including boilerplate removal).

In addition to building tools, we have developed a methodology for scholarly interactions with web archive data induced from our experiences with digital humanists and social scientists using our software. We call it the Filter–Aggregate–Visualize (FAV) cycle. The FAV cycle begins with a substantive question from a scholar who wishes to interrogate the web archive:

- **Filter.** Typically, scholars begin by focusing on a particular portion of the web archive, selected based on both metadata and content criteria. For example, the scholar might wish to focus on a particular range of crawl dates or pages from a particular domain (filtering based on metadata). Alternatively, the selection constraints might be content-based: e.g., pages that mention the phrase "global warming" or only pages that link to a particular site. The filtering criteria may be arbitrarily complex and nested: e.g., I am only interested in pages from 2012 from the Liberal Party that link to the Conservative Party and contain the phrase "Keystone XL".

- **Aggregate.** After selecting a subcollection of interest, scholars typically want to perform some type of aggregation that summarizes the data. Examples include counting, e.g., how many times each politician is mentioned, computing the maximum, e.g., find the page that has the most inbound links, or computing the average, e.g., how many times on average a particular politician is mentioned each day.

- **Visualize.** Finally, the aggregate data are presented in some sort of visualization, which could be as simple as a table of results or as complex as requiring an external application. Obviously, different visualization techniques are needed for different types of data.

Although each phase in the FAV cycle is equally important, the focus of our project thus far has been on the analytics (filter, aggregate). We have relied primarily on third-party tools for visualization, primarily as proofs of concepts.

Note that the phases in the FAV cycle are coupled in terms of the volume of data they generate. For example, the simplest "filter" would be no filter—i.e., the scholar is interested in the *entire* collection. Similarly, the simplest "aggregation" is no aggregation—i.e., list everything. This would, however, likely result in too much data to effectively visualize. On the other hand, the filter could be so specific as to match only a handful of records, in which case the "visualization" could be trivial—i.e., simply display the contents. The first case would correspond to what scholars call "distant reading" while the second corresponds to "close reading". It is our belief that tools should allow scholars to switch between these different modes seamlessly (and adjust the "distance" of their reading) as they interrogate the data.

Currently, there are three major categories of tasks that Warcbase supports:

1. Gathering page statistics.
2. Extracting and manipulating webgraphs.
3. Extracting and manipulating the textual content of pages.

In what follows, we provide three case studies that correspond to each of the above tasks, illustrating the FAV cycle.

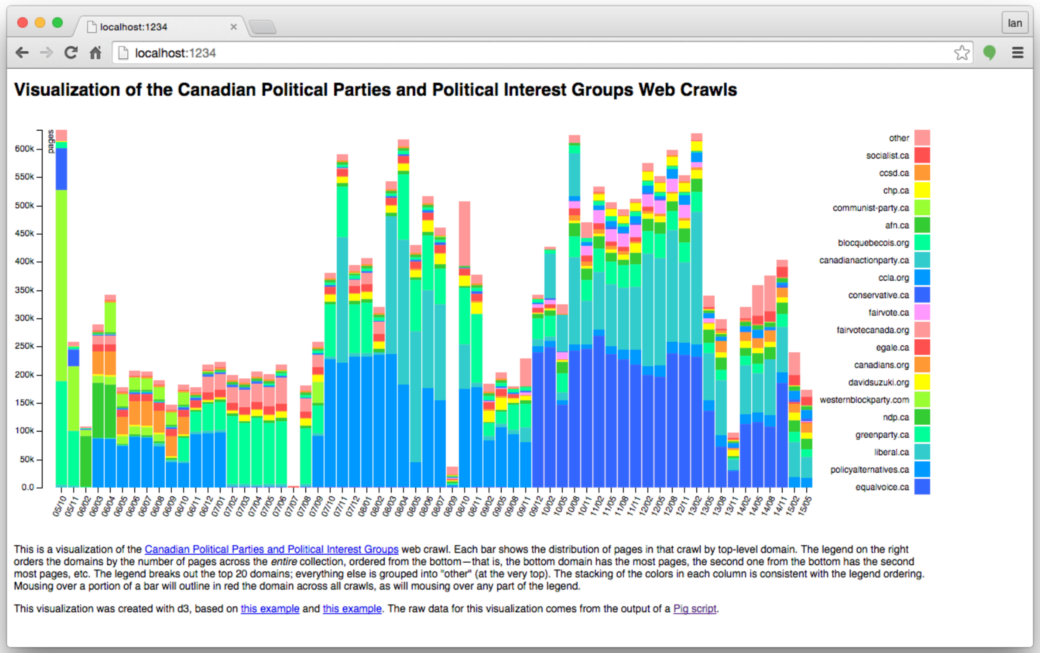
When scholars first approach a collection, they often have questions about collection statistics. For example, what sites are represented in the crawls? How many pages from each site? And across what periods of time? How much of a particular crawl (at a point in time) is occupied by pages from a particular site? Are the crawls capturing more or fewer pages from a particular site over time? Sites (i.e., domains) form the natural unit of aggregation because collection development is usually formulated in those terms. These exemplify the filter and aggregate steps in the FAV cycle described above.

To answer these questions, a scholar might write the Pig script shown in Figure 2 to analyze the Canadian Political Parties and Political Interest Groups collection, the output of which is a series of tuples consisting of (crawl date, domain, count), i.e., a particular crawl at a particular date captured so many pages from a particular domain. A brief explanation of the Pig script is provided below:

1. We begin by loading Warcbase classes and UDFs.
2. In the `arc`, `warc`, and `raw` lines, we load the ARC and WARC data (from HDFS).
3. In `a`, we keep only HTML pages and those pages that have crawl dates.
4. In `b`, we extract the crawl date and extract the domain using a UDF.
5. In `c`, `d`, we group by the domain and date, and count instances of pages belonging to each domain, date combination.

```
register 'target/warcbase-0.1.0-SNAPSHOT-fatjar.jar';

DEFINE ArcLoader org.warcbase.pig.ArcLoader();
DEFINE WarcLoader org.warcbase.pig.WarcLoader();
DEFINE ExtractTopLevelDomain org.warcbase.pig.piggybank.ExtractTopLevelDomain();

arc = load '/collections/CanadianPoliticalParties/arc/' using ArcLoader as
  (url: chararray, date: chararray, mime: chararray, content: bytearray);
warc = load '/collections/CanadianPoliticalParties/warc/' using WarcLoader as
  (url: chararray, date: chararray, mime: chararray, content: bytearray);
raw = union arc, warc;

a = filter raw by mime == 'text/html' and date is not null;
b = foreach a generate SUBSTRING(date, 0, 6) as date,
    REPLACE(ExtractTopLevelDomain(url), '^\\s*www\\.', '') as url;
c = group b by (date, url);
d = foreach c generate FLATTEN(group), COUNT(b) as count;
e = filter d by count > 10;
f = order e by $0, $1;

store f into 'cpp.sitecounts/';
```

Figure 2: Sample Pig script that generates site-level aggregate statistics by crawl date.



This is a visualization of the Canadian Political Parties and Political Interest Groups web crawl. Each bar shows the distribution of pages in that crawl by top-level domain. The legend on the right orders the domains by the number of pages across the *entire* collection, ordered from the bottom—that is, the bottom domain has the most pages, the second one from the bottom has the second most pages, etc. The legend breaks out the top 20 domains; everything else is grouped into "other" (at the very top). The stacking of the colors in each column is consistent with the legend ordering. Mousing over a portion of a bar will outline in red the domain across all crawls, as will mousing over any part of the legend.

This visualization was created with d3, based on this example and this example. The raw data for this visualization comes from the output of a Pig script.

Figure 3: Static screenshot of an interactive stacked barchart visualization of site-level aggregate statistics by crawl date, the output of the Pig Script in Figure 2. Each bar represents a particular crawl and shows the distribution of pages in that crawl by domain.

6. In `e`, we discard domains that have fewer than ten pages to reduce noise.
7. In `f`, we sort by date and domain.
8. Finally, we write the results to a file.

To facilitate visual analysis of results from this script, we have developed an interactive visualization with D3: a static screenshot is shown in Figure 3 but the visualization is available online.[9] In the stacked barchart visualization, each bar shows the distribution of pages in that crawl by domain. The legend on the right orders the domains by the number of pages across the entire collection, ordered from the bottom—that is, the bottom domain has the most pages, the second one from the bottom has the second most pages, etc. The legend breaks out the top 20 domains; everything else is grouped into "other" (at the very top). The stacking of the colors in each column is consistent with the legend ordering. Mousing over a portion of a bar will outline in red the domain across all crawls, as will mousing over any part of the legend. The code for this visualization is also part of Warcbase and can be adapted to explore arbitrary collections.[10]

Another common class of tasks that scholars often engage in is extracting the webgraph (i.e., the hyperlink structure between pages) from either the entire collection of a subset of the collection. A scholar might choose to focus at the granularity of an individual page (i.e., what does this page link to, what links to this page) or at the granularity of a site (domain) by aggregating all pages within a site. The scholar might be interested in a crawl from a particular period of time or the aggregate link structure across time. These are typical of the filter and aggregate steps in the FAV cycle we propose.

As a concrete example, the Pig script for extracting the site-link structure (i.e., hyperlinks aggregated at the site level) from the Canadian Political Parties and Political Interest Groups collection is shown Figure 4, which has a similar structure as the Pig script shown in Figure 2.

From this raw data, Ian Milligan has implemented a connector that allows the results to be visualized in the Gephi graph visualization platform.[11] A static screenshot is shown in Figure 5. Documentation of how to generate such visualizations is also included in Warcbase.[12]

Such a visualization effectively supports "distant reading" by scholars, who can focus on the macroscopic connections between sites as evidence addressing substantive questions. For example, in this static screenshot, the prominence of social media sites is readily apparent, which begins to illuminate their importance in modern political discourse. The interactive visualization also allows the scholar to zoom in on a particular neighborhood as an entry point to "close reading" in examining specific pages.

Beyond gathering collection statistics and examining the webgraph, the third main task that Warcbase supports via Pig is extraction of plain text from either the entire collection or a subset of the collection. Once again, the scholar is afforded the ability to select an arbitrary set of pages (i.e., filtering in the FAV cycle) and from those pages extract the plain text. We take advantage of Tika for removal of boilerplate such as navigation bars, page headers and footers, etc. The extracted output can then serve as input to off-the-shelf tools for topic modeling, generating word clouds, etc. The Pig scripts for accomplishing these analyses are quite similar to the other scripts presented above and therefore not shown for brevity. More details, however, can be found on the Warcbase wiki, which includes several sample scripts.

[9] `http://lintool.github.io/warcbase/vis/crawl-sites/`

[10] `https://github.com/lintool/warcbase/tree/master/vis/crawl-sites`

[11] `http://gephi.github.io/`

[12] `https://github.com/lintool/warcbase/wiki/Gephi:-Converting-Site-Link-Structure-into-Dynamic-Visualization`

```
register 'target/warcbase-0.1.0-SNAPSHOT-fatjar.jar';

DEFINE ArcLoader org.warcbase.pig.ArcLoader();
DEFINE WarcLoader org.warcbase.pig.WarcLoader();
DEFINE ExtractTopLevelDomain org.warcbase.pig.piggybank.ExtractTopLevelDomain();

arc = load '/collections/CanadianPoliticalParties/arc/' using ArcLoader as
  (url: chararray, date: chararray, mime: chararray, content: bytearray);
warc = load '/collections/CanadianPoliticalParties/warc/' using WarcLoader as
  (url: chararray, date: chararray, mime: chararray, content: bytearray);
raw = union arc, warc;

a = filter raw by mime == 'text/html' and date is not null;
b = foreach a generate SUBSTRING(date, 0, 6) as date, url,
    FLATTEN(ExtractLinks((chararray) content, url));
c = foreach b generate $0 as date,
    REPLACE(ExtractTopLevelDomain($1), '^\\s*www\\.', '') as source,
    REPLACE(ExtractTopLevelDomain($2), '^\\s*www\\.', '') as target;
d = group c by (date, source, target);
e = foreach d generate FLATTEN(group), COUNT(c) as count;
f = filter e by count > 10 and $1 != '' and $2 != '';
g = order f by date, source, target;

store g into 'cpp.sitelinks/';
```

Figure 4: Sample Pig script for extracting the site-link structure.

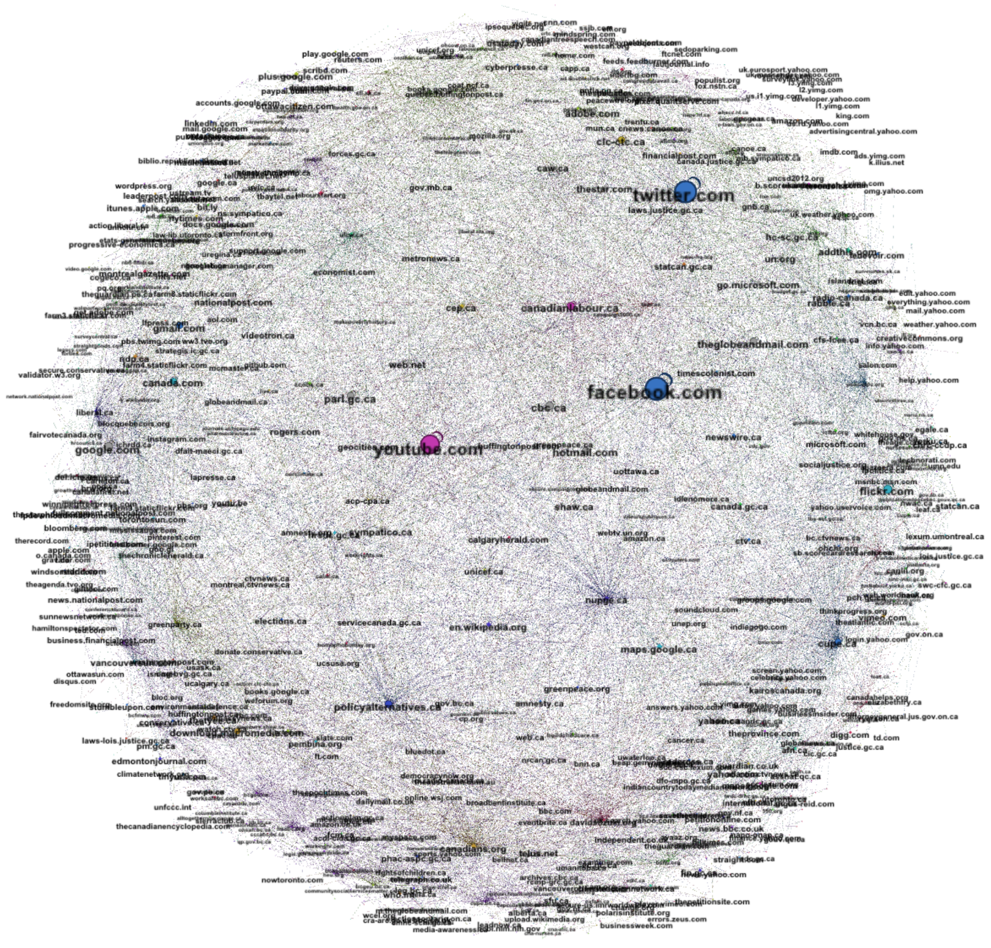


Figure 5: Screenshot from the Gephi graph visualization tool exploring the output of the Pig script shown in Figure 4.

3.4 Prototyping Full-Text Search Capabilities

In the project extension under the Web Archiving Incentive Program, we proposed to explore full-text search capabilities in Warcbase and the Hadoop software ecosystem. In particular, we enumerated three major challenges:

1. *Indexing infrastructure.* Given a collection of WARC and ARC files, how do we build the necessary index structures to support full-text search?

2. *Search infrastructure.* How does the user-facing search engine access the index structures to provide search capabilities?

3. *Search interface design.* What's the proper design of a search interface for web archives?

At the outset, we were well aware that adequately addressing all three challenges was far beyond the scope of the funding available for the project. Quite explicitly, we focused on an exploratory study to develop a proof of concept, as opposed to an effort to solve the problem in the general case. We furthermore sought opportunities to exploit and integrate existing open-source tools and packages, and to write code from scratch only when necessary.

In a collaboration with Ian Milligan and Nick Ruest (a digital assets librarian at York University), we have successfully deployed a full-text search tool over the entire Canadian Political Parties and Political Interest Groups collection. The search interface is available online at `http://webarchives.ca/`.

Our prototype integrates several existing open-source software components:

1. **Solr/Lucene.** Solr[13] is an open-source distributed full-text search system that uses the Lucene[14] search library at its core. Both are written in Java.

2. **The UK web archive's discovery tools.**[15] The UK web archive has developed a number of tools for exploring ARC and WARC data.

3. **The UK web archive's Shine interface.**[16] Shine provides a faceted search interface to indexes created by Solr/Lucene from web archive data.

We have successfully integrated the first two software packages into Warcbase. That is, we are able to build Lucene indexes over ARC and WARC data in a distributed manner (as a MapReduce job) using Warcbase commands. Detailed instructions for accomplishing this are available in the Warcbase wiki.[17] After these indexes have been built, they can be copied directly into an instance of the Shine search interface to provide full-text search capabilities.

A screenshot of our full-text search prototype is shown in Figure 6 for the query "recession". The main interface is similar to what most users today have come to expect from web search engines: a simple search box and results organized as an ordered list. Note, however, that the results are *not* algorithmically ranked, but simply presented in archival (i.e., temporal) order. Running down the left edge of the interface are controls for faceted navigation, which lets users filter content type (HTML, PDF, etc.), year of crawl, site, and a few other facets. Next to each facet we show the number of documents that match the filter criterion. We believe that facet navigation is sufficiently

[13]`http://lucene.apache.org/solr/`
[14]`http://lucene.apache.org/`
[15]`https://github.com/ukwa/webarchive-discovery`
[16]`https://github.com/ukwa/webarchive-discovery/tree/master/warc-discovery-shine`
[17]`https://github.com/lintool/warcbase/wiki/Building-Lucene-Indexes-Using-Hadoop`

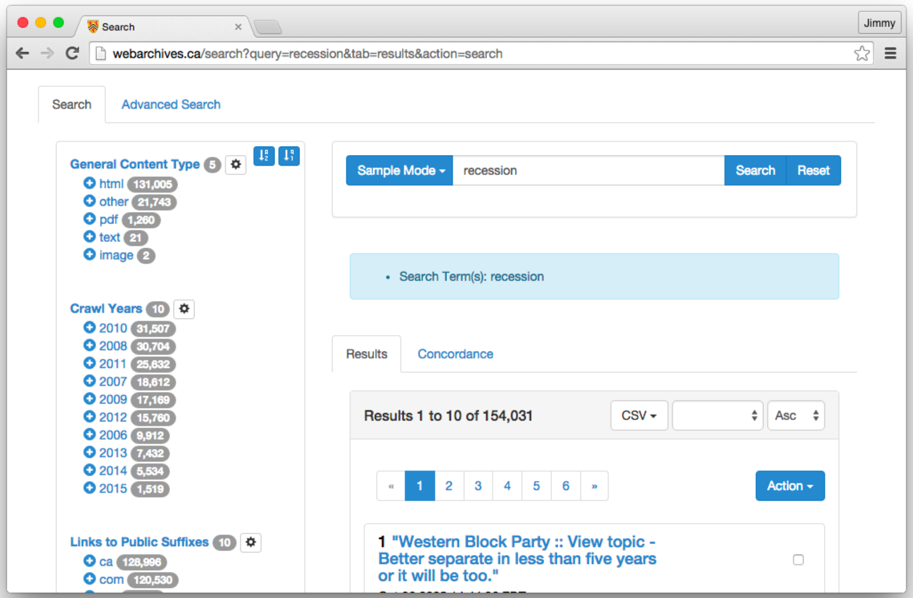
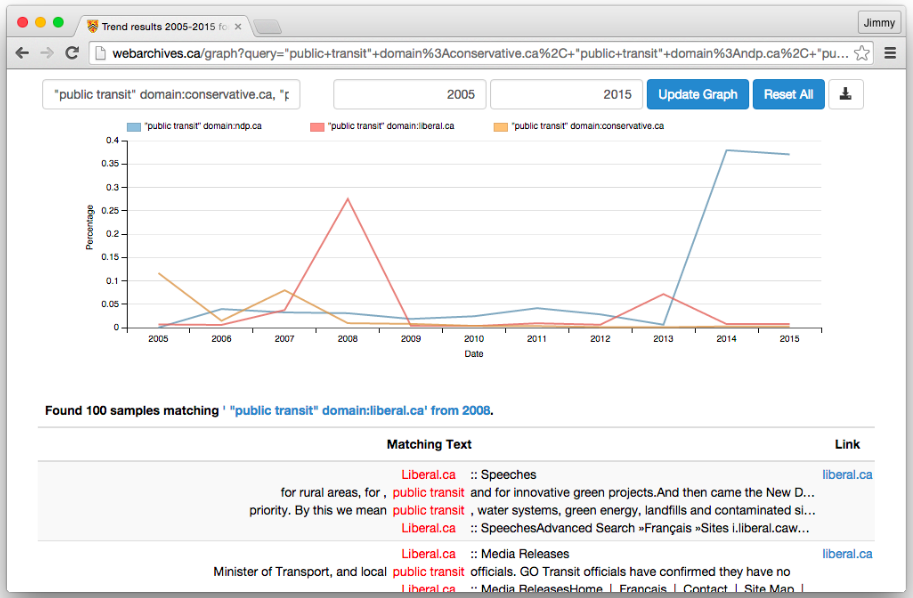


Figure 6: Screenshot of our prototype exploratory search interface for the Canadian Political Parties and Political Interest Groups collection.


Trend results 2005-2015
webarchives.ca/graph?query="public+transit"+domain%3Aconservative.ca%2C+"public+transit"+domain%3Andp.ca%2C+"pu…
Jimmy
"public transit" domain:conservative.ca, "p
2005
2015
Update Graph
Reset All
"public transit" domain:ndp.ca
"public transit" domain:liberal.ca
"public transit" domain:conservative.ca
Percentage
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
Date

Found 100 samples matching ' "public transit" domain:liberal.ca' from 2008.

Matching Text	Link
Liberal.ca :: Speeches for rural areas, for , public transit and for innovative green projects.And then came the New D… priority. By this we mean public transit , water systems, green energy, landfills and contaminated si… Liberal.ca :: SpeechesAdvanced Search »Français »Sites i.liberal.caw…	liberal.ca
Liberal.ca :: Media Releases Minister of Transport, and local public transit officials. GO Transit officials have confirmed they have no Liberal.ca :: Media ReleasesHome | Francais | Contact | Site Map |	liberal.ca

Figure 7: Screenshot of the trends visualization showing the prevalence of the phrase "public transit" on the websites of the Conservative Party, the Liberal Party, the New Democratic Party.

commonplace today (in sites like Amazon.com) that users will be able to manipulate the controls without requiring instructions or training.

As an alternative interface, Shine implements a "trends visualization" inspired by Google's Ngram Viewer,[18] shown in Figure 7. In this view, we are able to concurrently visualize the prevalence (i.e., frequency) of term matches over time for multiple queries. In this case, we show trends for the phrase "public transit" on the websites of the Conservative Party, the Liberal Party, and the New Democratic Party. The user can click on the line graph to obtain sample results, shown as a keyword-in-context concordance. In this example, a political scientist might use this interface to explore how each party's views on public transit evolve over time.

4 Ongoing Efforts

Although the funding provided by the Web Archiving Incentive Program has ended, there are two ongoing efforts to sustain Warcbase:

Development of Warcbase continues at the University of Waterloo via an undergraduate research assistant. Current efforts focus on Spark, a general-purpose analytics execution engine that has emerged as the replacement for MapReduce (and thus indirectly, Pig, since Pig "compiles down" to MapReduce). We are currently replacing Pig with Spark as the scripting language of choice for scholarly interactions with web archives. Spark promises several advantages, including better performance, tighter integration of language primitives with user-defined functions, as well as connections to a variety of "frontends" that would facilitate visualization. Fortunately, Spark is well integrated into the Hadoop ecosystem, which means that much of the Warcbase implementation will not need to be changed. There are sufficient similarities between Pig and Spark that we anticipate significant code reuse.

In collaboration with Ian Milligan (University of Waterloo), Nathalie Casemajor (Université du Québec en Outaouais), Matthew Weber (Rutgers University), and Nicholas Worby (University of Toronto), we have been awarded a grant by the Social Sciences and Humanities Research Council (SSHRC) of Canada to organize a hackathon around Warcbase, with the broader goal of developing some consensus around tools for analyzing web archives.[19] The event is planned for March 3–5, 2016 in the Robarts Library at the University of Toronto. We hope that this event will crystallize a sustainable community around Warcbase.

[18]`https://books.google.com/ngrams`
[19]`http://archivesunleashed.ca/`