

WebLibrary Design Progress Report

Nick Gerner

May 22, 2006

1 Introduction

This document presents the design principles underlying the WebLibrary project, plans and schemas following those principles, progress on the implementation of the production system, and results of some preliminary experiments using WebLibrary data. This document doesn't describe in-depth work on the user interface or the page store sub-systems.

The WebLibrary project provides a very large scale database interface to data from the Internet Archive, which has been archiving parts of the World Wide Web since 1996. Among the WebLibrary goals are to provide both a programmatic API to access data and computing resources, as well as a rich, web-based user interface to access a subset of those services.

Over the last year the following design principles have emerged as guidelines in decision making:

Lossless Crawler Data Storage

One, primary, goal of the WebLibrary is to faithfully store, and make available data from the Internet Archive. This means that data must be stored as it is received, without filters, and without processing which makes retrieval of the original data impossible. In some cases it may seem that the data ought to be transformed to make specific processing easier, faster, etc. This should be avoided, because there are uses of the data which cannot be foreseen at this time. We don't know all of the uses of the data and it would be a mistake to rule some of those out, without knowing that we're doing so.

Common Case Efficiency

Given the amount of data being processed, there are some decisions about how to handle the data to make some operations more efficient. While we don't know all of the uses of the data, there are some which we do know. We can speculate about which will be most common. When making decisions we should seek to make the most common cases the most efficient.

Automation, Insurmentation, Sanity Checks

We hope that this system will be run for many years, passing between several sets of hands. To that end we should automate processes where it makes sense, and insurment those processes so administrators can easily check the status of the system and its sub-systems. While the processes operate, sanity checks should verify results as they are produced.

2 History

Preliminary work on the WebLibrary began in Spring, 2005 including some work on synthetic data generation to supplement the small amount of data available from the Internet Archive at that time. During Fall, 2005 work on a processing pipeline began. This pipeline consists of: a data gathering Internet 2 connection to the Internet Archive, a preprocessing step, a database load step, and a user interface. Specifically, the initial connection to the Internet Archive was established, although slow and posing a number of problems; an initial version of the preprocessing application was written and tested. The database schema was also developed.

3 WebLibrary Meta Data

The WebLibrary database stores meta-data about World Wide Web pages. This data is collected by crawlers and served by the Internet Archive. Figure 1 illustrates web data as the crawler provides it, in a “Daisy Model”. Each “daisy” in this model represents a single URL. The petals of the daisy represent pages retrieved by the crawler. The petals are distinct because the data served from a URL might change from HTTP-request to HTTP-request. These variations might be due to changes by a human author over time, or the changes might be because of dynamic content (dates, ad links, etc.). The lines between petals and daisies represent web links.

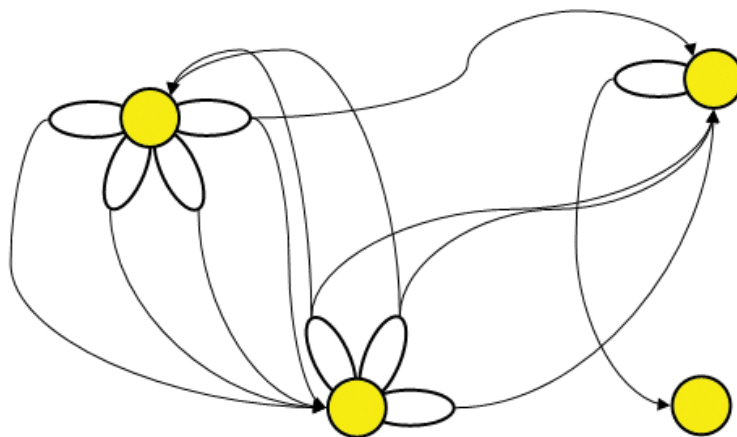


Figure 1: Daisy Model of Web Crawler Data

We present crawler data in this way, rather than as a simple graph, because this is the data the Internet Archive, and subsequently the WebLibrary, stores and serves. As more and more data is added to the WebLibrary an important feature of that data will be multiplicity of URLs. Each page-instance of that URL could be dramatically different (consider the changes of <http://www.cornell.edu> since 1996). Exploration of this topic is outside of the scope of this document. Indeed, this sort of exploration is what the WebLibrary is meant to support.

Figure 2 represents a simplified data diagram of the meta data as it is stored in the database. This

model corresponds to the Daisy Model presented in figure 1. This figure presents a database driven intuition of the crawler data. The actual schema, presented in figure A is an extension of this intuition. The development of this schema is described in an earlier report.

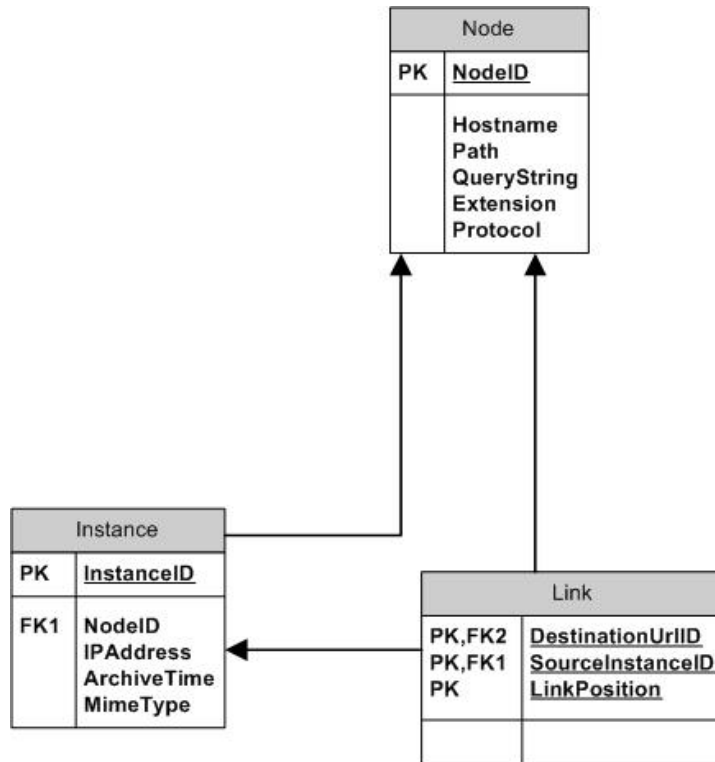


Figure 2: Meta Data Intuition

4 Data Pipeline Progress

4.1 Data Acquisition

Internet Archive data comes in sets called crawls. These crawls consist of hundreds of thousands of corresponding ARC and DAT files. ARC files contain compressed copies of archived web-pages. DAT files contain meta-data about these pages. Specifically, DAT files describe the page’s MIME type, archive time, etc. DAT files also explain the link structure of pages, including anchors (“a href”), images (“img src”), and scripts (“script src”). These files are stored at the Internet Archive in a massively distributed file system. Unfortunately, these files are not always well indexed. Comprehensive lists of constituent files for a given crawl are not always available. Often the file servers holding relevant files are known and file lists must be created by querying these servers. However, sometimes these servers go down.

Just prior to Spring, 2005 the data gathering connection and procedures to download large amounts of data were finalized. The data gathering process has been running smoothly over parallel FTP on the Internet 2 connection since the beginning of the semester. Most, if not all, of the EB crawl has

been retrieved and backed-up from the Internet Archive. This process has continued for the last semester without too many difficulties beyond the above issues about enumerating crawl files.

4.2 Preload

Before data can be loaded into a database, it must be preprocessed into data load files. To load data into most DBMS's (including Microsoft SQL Server 2000, which we use at this time), a load file must be created for each table, matching the schema of that table. Typically each line of the table file corresponds to one tuple and each field is delimited by some character, although fixed field and tuple lengths can also be specified.

For the WebLibrary, this means reading a DAT file sequence and producing several data load files, each corresponding to a table in the database. The challenge is to provide a very high throughput preprocessing application to do this. The files produced must be of an optimal size for loading into the database, or the preprocessed data must be directly loaded into the database directly from memory (if possible). The intermediate data produced by the preload process need not be archived, which gives some flexibility over this process. The files need not be documented or self describing. There need be no correspondence between the files.

The preliminary application developed in Fall, 2005 has continued to be used during Spring, 2006. This application has several shortcomings including little or no disk buffering, use of heavy-weight data structures, and an over-consistent output file set structure. This has limited throughput. We've continued to use this application while investigating other pipeline bottlenecks.

4.3 Dataload

Dataload has been one of the biggest areas of development during the Spring, 2006 semester. We had hoped that work from Fall, 2005 would have laid much of the experimental groundwork to begin production loading during Spring, 2006. And Spring, 2006 has seen approximately 10% of the EB crawl loaded into the database. However, a number of issues arose during the semester that slowed the process.

Fundamentally, this process is a straight forward load of preload output files into their corresponding tables in the database. There are a number of details that complicate this process. Many of these are detailed in the Fall, 2005 data load report and include duplicate elimination and data partitioning, but other issues have arisen.

One issue that has continually been a problem, and will continue to be one, is that the format of DAT files has not been extensively investigated with respect to data storage requirements. A simplistic view of DAT files is that they contain entries for pages and links to URLs. However, this glosses over the fact that the meta data for pages is rich and varied, and the fact that links come in many varieties with potentially different semantics. Some page and link entries contain a great deal of information that wasn't well understood at the time of the original meta-data schema development. Others are sparse and are missing important features. There are many types of links, and while some types are well understood, others are not. Because the uses of this data are not well understood either, it's important to remember the first design principle of the WebLibrary: lossless

data storage. The current schema may not be adequate to store this data, although it has been adapted (with some success) several times already.

Another issue has been the interface between the preload application and dataload. There are a great number of special cases in the DAT files (long field entries, unrecognized character encodings, use of delimiter characters, etc.). The preload application doesn't always handle these as special cases and dataload sometimes misbehaves in these cases, either crashing or corrupting data. The amount of data being loaded makes checking for corruption very difficult and expensive. Currently no automated sanity checks have been developed to verify the process. The current thinking is that the 10% of the EB crawl loaded can only be used experimentally and will have to be redone eventually.

A second issue with the preload-load interface is that of the disk. Currently, preload writes its output to disk in file sets. These file sets are internally consistent in the sense that each file in the file set corresponds with every other file in the fileset. The maximum size of any file in the fileset is guaranteed to be less than or equal to some maximum, externally set size. But all files in the fileset are closed when that maximum is achieved by any file. This consistency is not needed as these files are deleted after being loaded.

More importantly, the disk may not be needed as an interface between preload and load. It may be possible to load data directly from memory into the database. This could remove a big bottleneck in the preload-load process.

An issue which was not explored in-depth during Fall, 2005 is duplicate elimination. The data arriving from the Internet Archive contains many duplicates. At the Internet Archive the data is compressed and highly distributed. Storage capacity may not be as important a concern as it is for the WebLibrary (whose data is uncompressed and centralized). Duplicate elimination is important for storage-space concerns as well as correctness of selections. The current plan uses built-in DBMS duplicate elimination mechanisms. This is a bottleneck for the load process.

Figure 3 illustrates one proposal for external duplicate elimination. In this proposal, preload produces output with duplicates, although some limited, local duplicate elimination may be possible within preload (some preliminary results suggest that a great deal of duplicates are local). These output files could be pipelined or post-processed by a duplicate elimination process which removes duplicates with a set of load files. These duplicate eliminated results are then run through a final, global duplicate eliminator against the existing WebLibrary data before being loaded into the WebLibrary database.

4.4 User Interface

Figure 4 illustrates the user interface architecture designed over the last two semesters. This work has been lead by Blazej Kot; figure 4 is due to him.

The interface should provide both a programmatic API which can be used to write high-performance data analysis applications as well as a rich web interface which users can browse, select, and refine subsets of the data for off-line analysis.

BADLL is a library of data access functions for local use on or near the Scidata machine. This

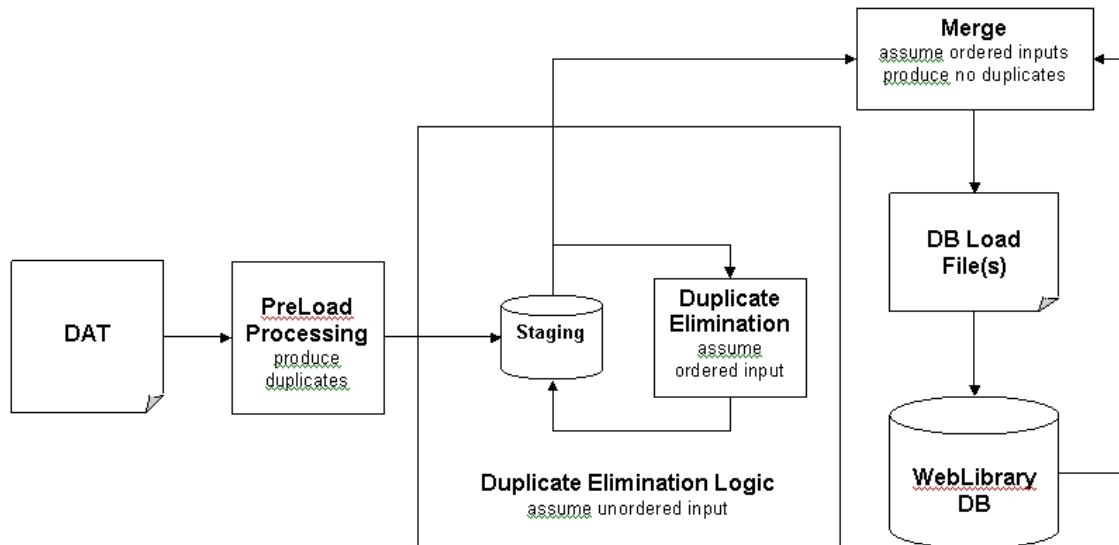


Figure 3: External Duplicate Elimination Proposal

library should connect directly to the MSSQL database and expose a number of functions to access data. There has been a lot of references to “subsetting” the data. The idea is that researchers are interested in part of the WebLibrary data and would frequently access these parts, without using the rest of the data. In order to provide fast, and safe, access to this data, researchers ought to be able to reference subsets which are longer-term groups of data they have selected (either by SQL or BADLL functionality). One idea was to provide a sandbox database to store these subsets independently from the rest of the WebLibrary data. These sandboxes provide a barrier protecting WebLibrary data while allowing researchers to more rapidly access and manipulate their data.

A second access path to the data is via a set of web services exposing some of the BADLL functionality. This makes the WebLibrary part of a service oriented architecture. Researchers can access WebLibrary resources (whether they by data or computational) from remote machines. Obviously, this access path should be authenticated, authorized and audited. The security implications of this (and any other) access path have not been thoroughly investigated.

Finally, researchers, and perhaps the public, should be able to access WebLibrary services via a web interface. At least two sets of functionality should be exposed in this way:

(1): WebLibrary data should be exposed via the web so researchers can access subsets or other selections of the data for simple download in some known format. Suggestions for this format have been simple comma or other delimited file or xml encoding. Either format could also be compressed at the server to limit bandwidth usage at both ends. This interface could also provide some monitoring support for researchers using WebLibrary resources (status of subsets, long running jobs, etc.)

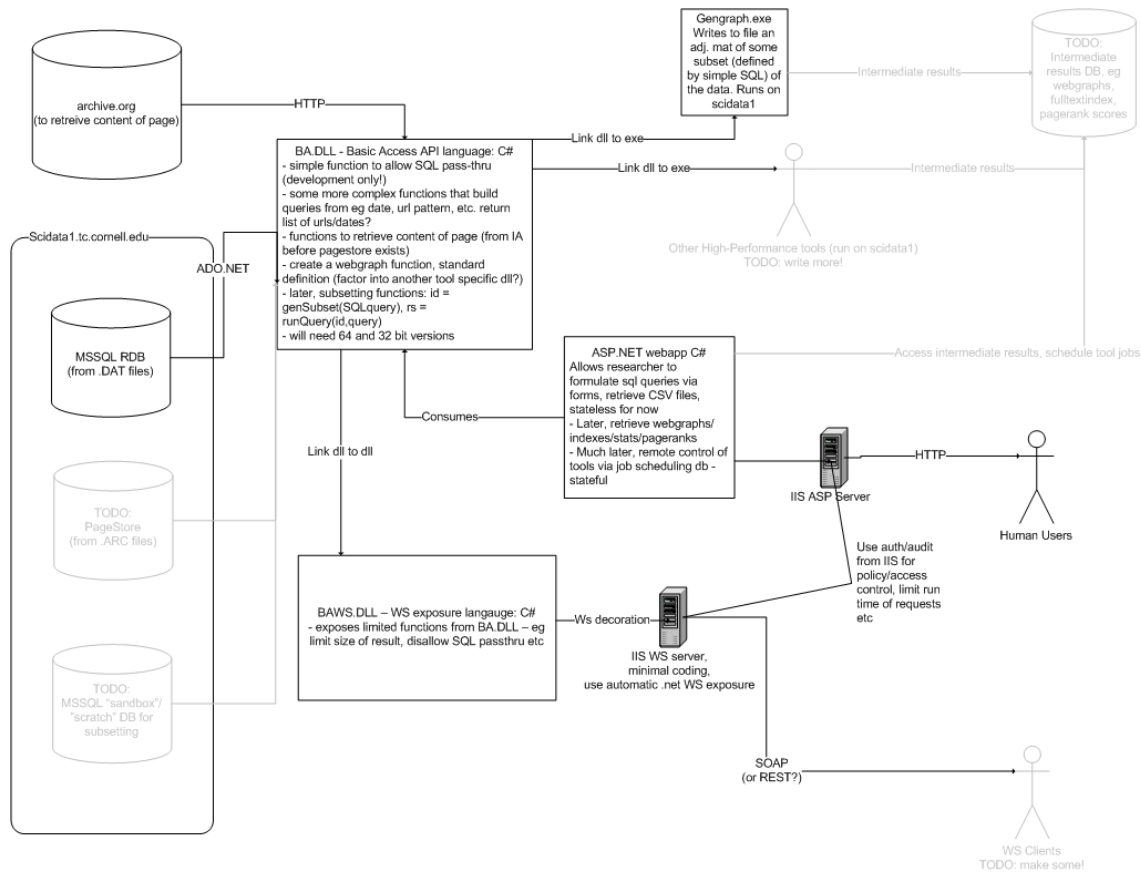


Figure 4: Interface Architecture

(2): The Retro Browser provides a web-browser like interface to weblab data. We're still in the early stages of development of this idea, but this interface is meant to provide a seamless browsing interface to the web, across time. Specifics of this interface are outside the scope of this paper, but this interface has a lot of mass appeal and should not be forgotten as we move forward.

5 Preliminary Usage Report

Pavel Dimetrievev, Blazej Kot, and I did a small study of web-graph analysis for Jon Klienberg during the second half of the Spring, 2006 semester. We used the WebLibrary data for some of the empirical part of our study. Given that little data of questionable quality was available we limited our use. However, we were able to get interesting results despite the issues described above. Most notably, we had to cope with the fact that only 10% of a single crawl was available. One thing we were able to do, was make a preliminary study of in and out degrees of the data. The specifics of this study are outside the scope of this document and available elsewhere. For example, we used the following query on a very small subset of the data running on a modern laptop computer. The results were almost instantaneous.

```

let nodes = SELECT distinct p.pageid, p.urlid
             FROM page p
             WHERE p.archive_time <= ALL
                   (SELECT p2.archive_time
                    FROM page p2
                    WHERE p2.urlid = p.urlid))

```

```

SELECT n.pageid, count(*)
FROM nodes n, link l
WHERE n.pageid = l.pageid
GROUP BY n.pageid

```

We also used the 10% data to do some sampling of data. Using Scidata1, we were able to extract 5000 complete URLs (concatenating the necessary fields) and save these for external processing almost instantaneously. Our external processing consisted of using an HTTP connection to the Internet Archive to do some link analysis across a three month period. This process was fast enough for our purposes. We were able to get link data about all 5000 URLs, across 3 months in a matter of hours. This provides an interesting baseline for performance comparison moving forward.

6 Open Issues

The following are a set of open issues as I see them. In general my feeling is that we've tried to move into a production mode. This is a very good goal, but I feel as if there are still many issues to investigate. While we may be able to load some data already, it would be useful to have a group of students or researchers who can investigate some of the open issues (and new issues as they arise) in parallel with this process. The Fall, 2005 semester saw a lot of experimentation and out-of-the-box thinking. In some cases this wasted time and resources as students made some well known mistakes or duplicated previous work. But a lot of issues were uncovered before they were problems and a lot of great solutions got developed as well.

- We need ARC and DAT file format and documentation (simple data-sheet describing formats, protocols, codecs, control characters, etc.). Currently we have discussions about these and it's clear that there is some understanding of this, but this knowledge isn't stored in any document that we can refer to later.
- We should investigate direct from memory data load from preload (if possible). This seems to hold so much promise. Although it might be impossible there hasn't been a strong argument as to why not except that we haven't tried.
- Rewrite preload application to avoid heavy-weight datastructures and use disk buffering. Load files need not be arranged into corresponding sets. This should not be a bottle neck. The application need only filter and de-multiplex the DAT files.
- Re-examine the meta-data schema with respect to DAT file structure across crawls. What issues arise when using the newest DAT files and the oldest? What is the schema of DAT

files? How has it varied over time? Are we missing any data? Are we grouping different data together (making separation of this data at a later time difficult or impossible)?

- Investigate memory-to-database load from preload into the database. MS SQLServer's ODBC driver supports this. Does ADO.NET have similar capabilities? If not, why not? And is the performance of ADO.NET worth the cost of writing to and reading from disk? When we run preload and load in parallel, will the disk be a point of contention?
- Investigate duplicate elimination. Be creative about this. The right solution might be built in and we may have to "live with it". But out-of-the-box thinking might solve this problem and push the bottleneck elsewhere.
- Security for the different access paths in the user interface needs to be thoroughly examined. What are the security requirements? What are the user scenarios we need to support and protected against?

7 Conclusion

The Spring, 2006 semester has seen a lot of production oriented work. This has been a move in the right overall direction. The WebLibrary is getting funding to provide the data for research consumption and to do so soon. In many ways this semester has seen a lot of victories. A preliminary 10% of the EB crawl available for some early work, and we've uncovered a lot of issues.

This document has tried to present some of the successes and problems we faced over the semester. What we've tried to do is express the open issues that need to be faced. Some of these are simple, but others require a little more investigation. The WebLibrary isn't at a 100% production level yet. This semester has been the first, of many, steps toward that 100% production level. What needs to be done is more work on solving some of these issues.

A Schema

