WebSnatcher:

Customized WWW Prefetching

Maria L. Gullickson, Ann L. Chervenak

January, 1998

1 Introduction

Magnetic disk prices are dropping and capacities are increasing rapidly. Future applications should be able to take advantage of this wealth of space to improve performance. WebSnatcher, as a part of the Personal Terabyte Project [3], does just this by prefetching large amounts of data from the World Wide Web. WebSnatcher is an application that automatically downloads Web pages that might be of interest to a user based on that user's profile. In a profile, users may specify explicit URLs (Uniform Resource Locators, the "addresses" of Web pages), a bookmarks file (such as that used by Netscape), and/or general areas of interest. WebSnatcher retrieves pages and stores them on a user's file system, allowing the user to specify how deeply hyperlinks are followed and creating an HTML (Hyper Text Markup Language, the data format used to encode Web pages) index of all pages retrieved to allow the user to navigate the results easily. Users may optionally archive any pages of interest to them, so that they can refer back to them in the future.

Network connections are not growing in speed as quickly as disks are growing in capacity. WebSnatcher prefetches data over the network to hide network latency. Users can configure WebSnatcher to run automatically at regular intervals. A user can access fast local storage to view the results of the latest execution of WebSnatcher.

This paper describes the design and implementation of WebSnatcher. First, we explain the features in WebSnatcher Version 1.0 and describe additional features planned or considered for later versions. Second, we describe in more detail several aspects of WebSnatcher, including the mechanism for fetching pages and how WebSnatcher constructs queries for the six search engines it currently supports. Next, we compare WebSnatcher with related applications. We conclude with an example of a user profile and sample results from executing WebSnatcher.

2 Features

2.1 Current Features

- Explicit URL Retrieval: The user may give WebSnatcher any number of explicit URLs to retrieve. WebSnatcher retrieves the pages corresponding to the URLs using the method described in section 3.1. It stores pages in a directory named "URLs" on the user's local file system.

- Areas of Interest: The user may specify any number of areas of interest. This is done by selecting a title and a list of keywords related to the interest. WebSnatcher retrieves pages for each area through the use of Internet search engines. This process is described in section 3.2. WebSnatcher stores pages for each interest in a directory named for the assigned title of that area.

- Customizable Search Engines: For searching on areas of interest, a user may specify one or more search engines. Currently, users may select from the following: Yahoo! [20], AltaVista [1], Lycos [14], Excite [6], HotBot [9], and Infoseek [11]. WebSnatcher sequentially submits queries to each specified search engine and saves the top matches for each engine.

- Bookmark Retrieval: The user may specify the location of a bookmarks file, and WebSnatcher will retrieve all pages referenced in this file. The program opens the bookmarks file and parses it to find URLs, which are indicated by "HREF" labels in HTML. WebSnatcher then fetches the page corresponding to each URL and stores the pages in a directory named "Bookmarks".

- Full Page Retrieval: WebSnatcher will not only retrieve an HTML file, but also any images and frames contained in it that are addressed "relatively" (e.g., "images/bob.gif" as opposed to absolute links such as "http://www.acme.com/images/bob.gif"). If all such links are relative to the current page, the a user can view the entire page from local storage.

- Customizable Retrieval Depth: WebSnatcher can also retrieve any pages referenced within the main page that are addressed relatively so that hyperlinks may be followed normally. The user may specify the depth to which hyperlinks are retrieved, and may even specify a different depth value for each item in the profile.

- HTML Index: WebSnatcher, once done retrieving Web pages, creates an HTML file listing all main pages retrieved with hyperlinks to them. The HTML titles of the retrieved pages are determined by parsing each page. The index file includes both the title and original URL, along with information about how much data was retrieved for each category and total (in bytes).

  In the index, pages are listed by category of retrieval. Within each search category, URLs are listed by search engine. This also allows users to see which engines are returning which results, so that they can choose appropriate search engines for future queries. For each search engine, URLs are in order of closeness of match to indicate which pages are likely to be most useful.

- Data Clean-up: WebSnatcher automatically removes all files from its last run, if any, before retrieving new files. A separate executable is also included so the the user may clean up old files without retrieving any new ones. The purpose of this clean-up is to avoid saving stale copies of data that are unlikely to be accessed again.

  When WebSnatcher runs, it creates a bookkeeping file listing all main pages retrieved and what depth each has been retrieved to. During clean-up, this bookkeeping file is read, and WebSnatcher removes pages in the reverse order of their creation. First, it removes hyperlinks within a page recursively to the appropriate depth; then it removes frames similarly; next, it removes images; and finally it removes the page itself. As directories that were created become empty, it removes them as well.

- Selective Archival: By default, WebSnatcher removes pages retrieved by earlier runs of the application before fetching updated pages. In some cases, users may want to save Web pages that WebSnatcher retrieved previously; WebSnatcher includes the capability to archive pages on the user's file system.

  Users may archive specific pages by selecting the appropriate URL from a list; WebSnatcher will copy that file and all its auxiliary files to a directory called "Archive". Within this directory, files are separated by date, so that a user can archive multiple versions of the same page without conflict. An index to archived files, "archive.html", is created to allow navigation of these files.

  Users may later choose to remove all pages archived before a certain date. This removal process is very similar to the clean-up process described above. When pages are removed from the archive, the index is updated accordingly, so that it always contains an accurate representation of the archived data.

- Text Interface to Profile: A text interface for maintaining a user profile is provided, allowing the user to create a new profile or add, remove, or modify elements of an existing profile.

  This interface initially looks for a file containing the user's profile. If the file does not exist, it prompts the user to create a new profile. If the file does exist, the profile program reads the information into a structure containing all necessary data. It then allows the user to view and modify the profile data. Users can also run WebSnatcher directly from this interface.

- Timed Runs: We expect that the user will want to run this program regularly (for example, every night). A customizable perl script that allows a user to specify the frequency and time of day for application execution is included; this script runs WebSnatcher automatically.

The script determines how long until the first run and sleeps until that time. Then it runs WebSnatcher, determines the time until the next run, and sleeps until that time, when it wakes up and runs again. It repeats this indefinitely.

2.2 Planned Features

- Graphical Interface to Profile: We will implement a more sophisticated, graphical interface to the profile that is easier to use.

- Anycasting [2] Retrieval: The Anycasting project at Georgia Tech allows the user to specify a set of servers from which a request can be satisfied. WebSnatcher will evaluate the servers, keeping some response time data during fetching and using past performance information to guide selection of a server. This is useful if a user wants to get some data (such as weather forecasts or stock quotes), but is not concerned with which of several servers provide the information. WebSnatcher can automatically choose the one that will likely incur the smallest load on the network and return the data fastest. This feature would be especially helpful for pages that are updated frequently, such as news servers and stock market servers.

2.3 Possible Features

- Variable Run Timing: A user may want to specify different schedules for different components of their profile. Currently, WebSnatcher is set up to retrieve pages corresponding to every part of the profile sequentially in one run. To allow variable scheduling for different elements of the profile, we will have to separate the code for each item, so that the perl script can invoke only one portion of the code at a time.

- Page Exclusion: Often a search will return a page that is of no interest to the user. In this case, there is no need to repeatedly download and store that page. A user should be able to specify that a given URL is not of interest. On future searches in that area, WebSnatcher would not prefetch that page.

- Advanced Searches: Most search engines allow advanced search techniques such as whole phrase matching, searching only in titles, or specifying words to exclude. Users may want to specify such advanced queries in WebSnatcher. Since each search engine implements these differently, separate code must be written for each search engine. In addition, not all search engines allow the same advanced search. For example, not all search engines provide a means to exclude matches on particular words.

- "Cool" Lists: Some sites provide changing lists of "What's Cool", "What's New", or other pages of general interest. Letting users choose among such lists and add list pointers to their WebSnatcher profiles would be straightforward.

- Detection of Modification: If WebSnatcher could determine whether a Web page has been modified, it could eliminate unnecessary fetches and only retrieve Web pages modified since they were last retrieved. There is, however, some difficulty in determining if a Web page has changed without first downloading it for comparison. Another useful feature would be to to notify users when pages change, via e-mail, a graphical flag, or some other means.

- Customizable Number of Query Returns: A user may want to have more control over how much data WebSnatcher prefetches to their file system. Allowing the user to specify a maximum number of pages to return for each query would give them this control. There may be difficulty, however, choosing the "best" pages to return when merging query results from multiple search engines.

- Usage Monitoring: WebSnatcher could monitor how often each category of pages is referred to by the user and set its own schedule of retrieval based on this. If a user's habits changed, WebSnatcher could detect this automatically. Such monitoring is currently beyond the scope of the WebSnatcher project.

3 Implementation

3.1 Web Page Retrieval

Either by explicitly naming URLs, by including a bookmarks file, or by performing a search engine query, the user's profile generates a list of URLs. In this section, we discuss the mechanism WebSnatcher uses to fetch the corresponding pages.

First, WebSnatcher must parse the URL to find the host and the page location. For example, given the URL "http://www.prism.gatech.edu/~gt8572b/index.html", the host is "www.prism.gatech.edu" and the page location is "~gt8572b/index.html". Once the host is known, WebSnatcher initiates a TCP/IP socket connection with the host on port 80, which is the standard port for HTTP. In some cases the URL includes an alternative port specification. For example, if the URL is "http://www.prism.gatech.edu:8888/~gt8572b/index.html", then this machine uses port 8888 as its HTTP port. WebSnatcher must parse the alternative port out of the URL, and then initiate the socket connection on this port, rather than on port 80.

Next WebSnatcher sends a message on this socket requesting the given page. The format of this message is fairly simple. To retrieve a file, WebSnatcher sends a message of the form "GET <file path> \n\n". WebSnatcher then reads the requested page from the socket and stores it to a file in the appropriate directory. It forms file names for the pages from the URLs, with '/'s replaced by ':'s to avoid confusing filenames with local directories. For example, if WebSnatcher retrieved the page "http://www.acme.com/bob.html", it would be stored to a file named "www.acme.com:bob.html".

Once a page is retrieved, WebSnatcher reads through the HTML looking for references to images and retrieving each image addressed by a link relative to the current directory. (If links specify absolute URLs, WebSnatcher does not retrieve them.) Then WebSnatcher looks for references to frames and recursively retrieves any relatively addressed frames, just as it retrieved the original page. (This way images within frames or frames within frames are also retrieved properly.) WebSnatcher stores these auxiliary files (images and frames) in the appropriate position relative to the main page with their original file name, maintaining the server's directory structure on the user's file system.

Finally, WebSnatcher closes the socket. The user's file system now contains a local copy of the requested page and its auxiliary files.

Each time WebSnatcher retrieves a Web page, a depth value is associated with it. The initial value is a non-negative integer chosen by the user. On a retrieval, if the depth value of a page is greater than 0, WebSnatcher parses the page looking for hyperlinks. It recursively retrieves any hyperlinks found using relative addresses, decrementing the depth value by one. This continues until the depth value reaches 0.

3.2 Interest Searching and Retrieval

One section of the user profile contains sets of keywords related to user interests. To search on a given interest, WebSnatcher submits queries to one or more search engines. It submits these queries by creating a URL containing the keywords of a search; retrieving the query URL initiates the search on the remote search engine. This URL is retrieved just like any other URL, as described in section 3.1. Then WebSnatcher parses the returned HTML page to extract URLs corresponding to search results (if any). Finally, it retrieves these pages and store them on the local file system.

The URL created to perform the search and the format of the result vary from engine to engine, so WebSnatcher includes engine-specific code for these tasks. We determined the formats of the URLs and the search results by examining the queries and results visible through the Netscape Web browser.

Once a list of URLs from all selected search engines is compiled, the list is checked for duplicates and these are removed. (It is not uncommon for a single search engine to return the same page more than once, and certainly when using more than one search engine the problem is even more likely to occur.) This ensures that WebSnatcher does not waste time retrieving the same page and all its components multiple times.

In the sections that follow, we discuss the six search engines which WebSnatcher supports, including unique features of their query structure and search results.

3.2.1 Yahoo!

Yahoo! stores information by category in a custom-made database, which is not publicly available. Sites are added to the database mainly as entered by users, but search robots locate some as well. [20]

In Yahoo!, the URL from which to retrieve search results based on keywords WORD1, WORD2, . . . WORDn takes the form:

http://av.yahoo.com/bin/search?p=WORD1+WORD2+ . . . +WORDn.

When parsing the results of a search, we first check to see if there is an indicator that no pages were found. In Yahoo!, this indicator is the string "Sorry, no matches were found". If matches were found, they are the only list items within the page. In HTML, list items follow the tag "<li>", so WebSnatcher simply searches for these and extracts the next URL, which is the next substring beginning with "http".

3.2.2 AltaVista

AltaVista's Web index contains about 200 gigabytes of data and completes searches in under a second. Scooter, AltaVista's Web spider, creates the index, locating about six million pages per day. AltaVista indexes every word within the pages for searching, along with information about the location of the word within the document. [1]

The URL used to retrieve search results in AltaVista should be of the form:

http://altavista.digital.com/cgi-bin/query?pg=q&what=web&fmt=c&q=WORD1+WORD2+ . . . WORDn.

The phrase "No documents match the query" is the alert that no matches were found by the AltaVista search engine. If there were documents found, they are listed as preformatted text, so WebSnatcher searches for the appropriate tag, "<pre>". Within this area, matches follow a period and are contained within "<a>" tags, so WebSnatcher looks for the URL following each string ".<a".

3.2.3 Lycos

Lycos includes a large database, covering most every URL on the Web. [17] Searches are very fast. [8] Users may enter their sites into the Lycos database. Afterwards, Lycos will visit the site and follow hyperlinks within it to find more sites to index. [14]

Search results from Lycos are found by retrieving URLs of the form:

> http://lycospro.lycos.com/cgi-bin/pursuit?mtemp=nojava&etemp=error_nojava&at=lycos&
>
> npl=matchmode%3Dand%26adv%3D1&query=WORD1+WORD2+ . . . WORDn.

"The following search word(s) were not" is the string which indicates that Lycos was unable to find any results from the search. If it was able to find results, they will be the only hyperlinks in the document, so one can simply search for "<a" tags and take the URLs from there.

3.2.4 Excite

Excite searches return results that are related to the ideas of the keywords, not merely documents that contain the terms entered. It does this by examining indexed documents to determine what terms or phrases are related to each other and searching for these related terms in addition to the specified terms. Excite gives users access to more than 50 million Web pages. [6]

In Excite, search results come from a URL with the form:

> http://www.excite.com/search.gw?c=web&FT_1=w&FL_1=3&FI_1=WORD1+WORD2+
>
> . . . +WORDn&FT_2=w&FL_2=4&FT_3=2&FI_3=&move=advanced&numFields=3&lk=
>
> default&sort=relevance&showSummary=false&perPage=20&advanced=Search.

This URL also contains information to indicate that the full descriptions need not be returned, only URLs, and that up to 20 matches should be returned.

URLs of pages satisfying the search results are found as the first URL following bold text ending with a percent sign, so WebSnatcher looks for URLs following the string "% </B>". If this string is not present, no pages were found.

3.2.5 HotBot

HotBot uses a Web crawler that covers the entire Web every two weeks. It has been explicitly designed to keep up with the fast-growing Web. This scalability is allowed through the use of NOW (network of workstations) technology. [9]

The URL to retrieve results of a HotBot search takes the form:

> http://www.search.hotbot.com/IU0nfYOl7C5920270C63179C3443417A359D856F/
>
> hResult.html/?SM=MC&MT=WORD1+WORD2+. . . +WORDn&DV=7&RG=.com&DC=
>
> 25&DE=0&OPs=MDRTP&_v=2&DU=days&SW=web.

A lack of results is indicated by the string "Sorry– your search yielded no results." If there are results, they can be found as the first URL following bold text ending with a period, i.e. following each string ". </B>".

3.2.6 Infoseek

Sites must be explicitly added to Infoseek, as it does not crawl the Web looking for pages to add. [11] Searching on Infoseek is very fast. [8]

Infoseek returns search results for URLs of the form:

> http://www.infoseek.com/Titles?qt=WORD1+WORD2+. . . WORDn&col=WW&sv=IS&
>
> lk=ip-noframes&ud4=1&nh=20.

The presence of the substring "Infoseek found no results for:" indicates that no pages were found from the search. If this is not the case, we proceed to look for the URLs. Each HREF pertaining to a search result is within bold tags, so we can simply look for the string "<b><a" and then proceed to the next "href=" and parse out the URL.

3.3 Planned Improvements

As mentioned above, when WebSnatcher retrieves a main Web page, it retrieves auxiliary files along with it; these file include images, frames, and possibly hyperlinks. WebSnatcher only retrieves those files addressed relative to the current page's location rather than absolutely.

WebSnatcher then places these extra components in the appropriate location relative to the original page. Thus, if WebSnatcher retrieves a file referred to as “images/bob.gif”, it creates a directory called “images” in the directory that contains the main page and places bob.gif in this new directory. This means that if a file path goes up a directory (e.g. “../bob.gif”), it is placed in the parent directory. In theory, there could be so many references to parent directories that we get to the root directory and cannot go any higher to create the file. This is extremely unlikely and has never occurred in practice. Because WebSnatcher duplicates the directory structure of the original documents on the local file system, the WebSnatcher user cannot predict where it will store files. An alternative is for WebSnatcher to choose the directories for the auxiliary files on the local file system; WebSnatcher would then change the HTML references in the retrieved pages to reflect the new locations of the auxiliary files. This second scheme, which we do not currently implement, would require altering the retrieved HTML files, but would simplify file management.

WebSnatcher does not retrieve absolutely addressed files (file references that include a complete URL), since their path requires that a Web browser go over the Web to download the file, rather than looking on the local file system. It would be possible to download these files onto the local hard drive, but again, this would require changing the original HTML to refer to the new location.

Which auxilliary files are retrieved and where they are placed are two issues we continuing to study. The trade off between organization and speed of access will be weighed further to determine whether a change in retrieval and storage is warranted.

4 Related Work

Various applications perform similar functions to WebSnatcher. In this section, we compare WebSnatcher with several of these related applications.

4.1 Syskill & Webert

Pazzani, Muramatsu and Billsus at the University of California, Irvine, have developed an agent called Syskill & Webert [16]. Syskill & Webert allows a user to rate the interest level of various pages on a specific topic and, from these ratings, creates a profile based on an analysis of terms in the documents. Based on this profile, the application suggests which hyperlinks a user might like to follow from the document currently being viewed, by adding graphical icons to the page in the user's Web browser indicating the expected interest level. It can also construct a Lycos [14] query to find other pages that would interest a user.

This application requires that the user's machine download all pages for analysis first, and if a user selects a certain page, they must load the page again for viewing, possibly from a cached copy. However, prefetching of hyperlinks can be done while the user is still viewing the current page, to reduce delays. Users can choose pages based on their predicted interest level and potentially avoid fetching uninteresting pages. WebSnatcher requires users to check for themselves which pages are useful, but allows them to do this from fast local storage.

4.2 Loon and Bharghavan

Tong Sau Loon and Vaduvur Bharghavan have developed a system to deal with the latency and bandwidth problems in Web browsing [13]. Their work involves three techniques. First, the system maintains a profile of user access patterns and group access patterns. Users may choose which groups to join based on their interests. Second, the application filters HTTP requests and responses to reduce data transmission. For example, color depth of images may be reduced, or HTTP request headers may be reduced. Third, the system hoardes documents based on user profiles. Access patterns will be used in combination with the current document request to determine which documents the user is likely to request in the near future. The application prefetches these documents to a user's machine. This not only alleviates latency and bandwidth problems, but allows users to continue browsing should they become disconnected from the network for a period.

This is somewhat similar to our work; it involves maintaining a user profile and prefetching

Web pages based on that profile. The profile is more sophistocated, in that information is automatically learned from past usage patterns. WebSnatcher uses a more static profile, which users modify directly when it is not in use.

4.3 Recommendation Systems

Recommendation systems allow users to join groups based on their interests [5]. For each group, the system keeps track of people's preferences for various pages. Users explicitly tell the system what pages they like and dislike. The system can then determine which users have similar interests and make further recommendations to a user based on how other users have responded. There are several systems of this sort available today, including Firefly [7], Net Perceptions [15], LikeMinds [12], and WiseWire [19].

By contrast, WebSnatcher is a single-user tool, and so does not incorporate the reactions of other users in selecting sites, but the idea is rather intriguing. The goal of such systems s to give users not only information about Web sites in their areas of interest, but also pointers to sites that are likely to be interesting to them.

4.4 Wget

Wget [18] is a GNU utility to retrieve Web pages. It only works given specific URLs, not areas of interest, but allows users to select which hyperlinks it follows in a variety of ways. Users can choose whether to follow all hyperlinks, only relative hyperlinks, only hyperlinks referring to the same host, or only hyperlinks in a given domain. Users may also specify file types they want retrieved by listing suffixes or patterns; when downloading, Wget either includes or excludes files whose names contain those patterns. Similarly, users can include or exclude files in specific directories or specify that no files that refer to the parent directory are allowed.

These options allow the user to restrict the data they are retrieving. An application like WebSnatcher can be slow and use a good bit of disk space. Using Wget, the user can limit the amount of data being retrieved without eliminating pages of greatest interest.

4.5 The Informant

The Informant [10] is a Web notification service run by members of the Computer Engineering Group at Dartmouth College. With the Informant, users log on to a Web site, where they can select up to five URLs and up to three sets of keywords. The Informant searches on AltaVista [1], Lycos [14], Excite [6], and Infoseek [11], combining the results to determine the ten most relevant sites. When any site is updated or if a new page appears in the top ten list, an email notice is sent to the user, who can then log into the Informant and visit the new sites. Users may choose how often it checks sites for changes: every 3, 7, 14, 30, or 60 days.

The Informant determines that a page has changed by computing a hash value for the page and comparing it to the page's previous hash value. In this way merely the hash values need to be stored for comparison, rather than the entire page. A central server must download the entire page to compute the hash value for comparison, but users only need to wait for the download time if there is something new for them to see.

The creators of the Informant have several ideas for improvements which they plan to implement. They would like to allow users to keep track of more than ten sites per query. The Informant will include more search engines to make the results more reliable. There are also plans to allow users to mark particular sites as irrelevant so that they are not repeatedly returned as matches.

The Informant lets users know when pages have been updated; when this occurs, users must fetch the changed pages one at a time over the network. By contrast, WebSnatcher stores pages on the user's local file system; it does not currently notify users when pages change, but we hope to add such functionality.

4.6 HPP

HPP [4] is an extension to HTML which allows document creators to specify dynamic versus static sections of a document. In this way, users can cache the static sections and only need to repeatedly retrieve dynamic sections. For some applications, executing code will produce HTML documents. With HPP, this code could be made more efficient because it could simply produce the dynamic portion of the document, rather than the entire document.

WebSnatcher could make use of HPP to repeatedly fetch only the dynamic portions of pages (for example, the results of search engine queries). WebSnatcher might also use HPP to decide that dynamic sections of certain HTML documents change too frequently (such as stock and news services) to make prefetching useful, and choose not to download those pages.

5 Example Profile and Results

To get a better idea of just how WebSnatcher works, we will provide an example profile and explain the results produced. Consider the profile below:

```
Your profile contains the following URLs:
1 - http://www.intellicast.com/weather/atl/ (2)
2 - http://sandbox.99x.com/noshock/freeload.html (2)

Your profile contains the following search engines:
1 - Infoseek
2 - AltaVista

Your profile contains the following interests:
1 - cow (1): bovine farm animal

Your profile contains the following bookmarks file:
/net/hu4/maria/.netscape/bookmarks.html (3)
```

First we will explain the profile. The number in parentheses following each item indicates the depth to which hyperlinks are followed and retrieved. This profile contains 2 explicit URLs (http://www.intellicast.com/weather/atl/ and http://sandbox.99x.com/noshock/freeload.html), both of which have a retrieval depth of 2. This means that the main page is retrieved, any local hyperlinks within it are retrieved, and any local hyperlinks within those pages are retrieved. In addition, any images or frames within any of these pages are retrieved.

Searching for areas of interest will be done using two search engines: Infoseek and AltaVista. The user has specified one area of interest: "cow". Searching is done using the keywords "cow", "bovine", "farm", and "animal" on both search engines and resulting pages are retrieved to a depth of one.

Finally, a bookmarks file is specified (/net/hu4/maria/.netscape/bookmarks.html). This bookmarks file has references to the following URLs:

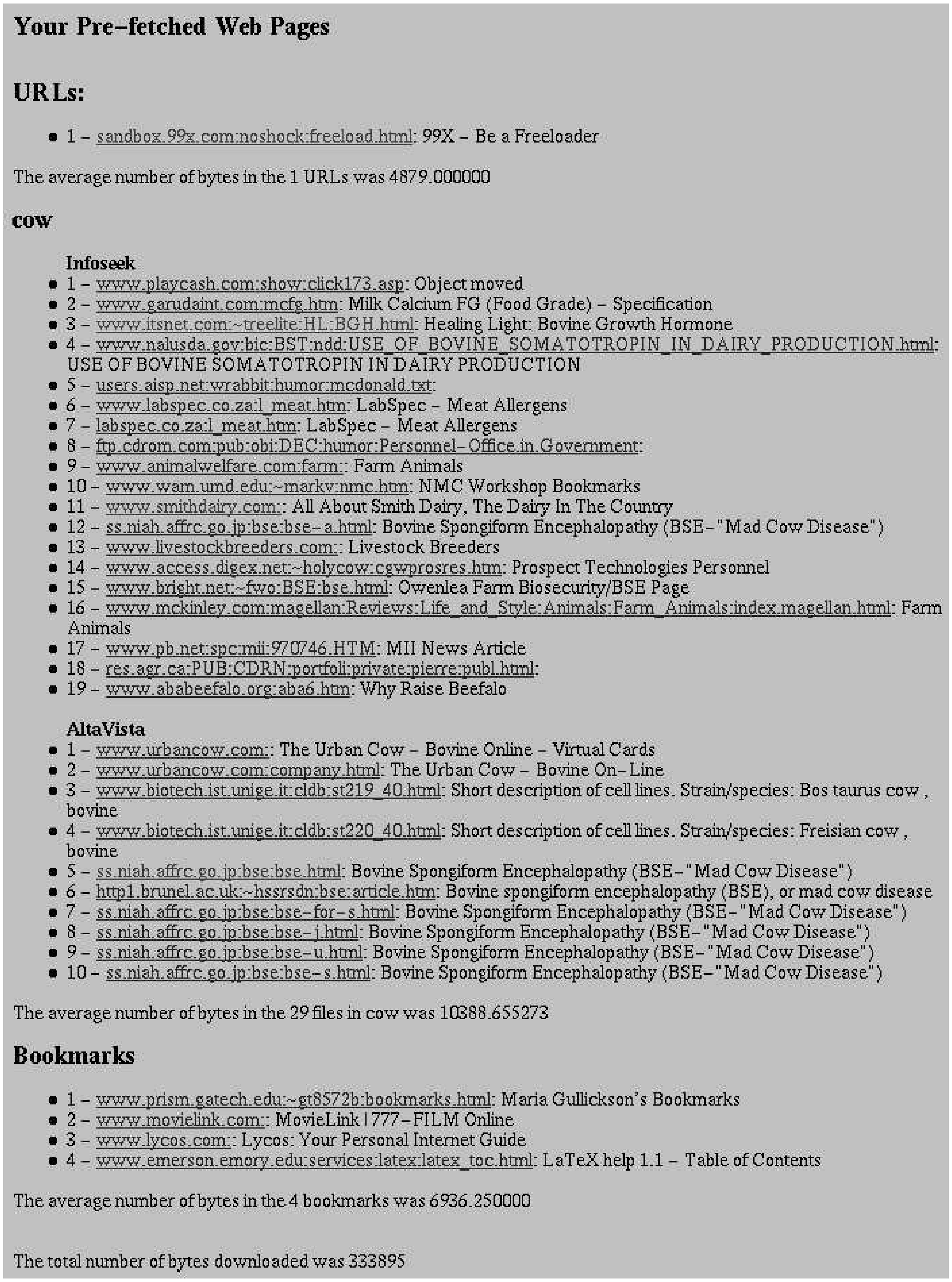
- http://www.prism.gatech.edu/~gt8572b/bookmarks.html
- http://www.movielink.com/
- http://www.lycos.com/
- http://www.emerson.emory.edu/services/latex/latex_toc.html

Each of these are retrieved with hyperlinks followed to a depth of three.

Upon retrieving all of these pages, WebSnatcher creates an HTML file with hyperlinks to each found page. Figure 1 shows the HTML page for our example profile. The page includes hyperlinks to each of the main pages retrieved, listed by category. Areas of interest are further broken down by search engine. For each item listed, the page includes the URL of the original document and the title of the document (or "<Untitled>" if none is provided). For each category, there is a summary of the amount of data collected (total number of main pages and average bytes per page). Finally, there is a total of the bytes received.

6 Conclusions

We have described WebSnatcher, an application that prefetches Web pages and stores them on a local file system based on a user profile. Users can specify explicit URLs, the location of a bookmarks file, and areas of interest for searching. Along with areas for searching, users may select one or more of the supported search engines to use in making queries. Users can also specify how deeply it follows hyperlinks within the page. A perl script is included to allow users to run WebSnatcher automatically at periodic intervals. WebSnatcher automatically cleans up



Your Pre-fetched Web Pages

URLs:

- 1 – sandbox.99x.com:noshock:freeload.html: 99X – Be a Freeloader

The average number of bytes in the 1 URLs was 4879.000000

cow

Infoseek

- 1 – www.playcash.com:show:click173.asp: Object moved
- 2 – www.garudaint.com:mcfg.htm: Milk Calcium FG (Food Grade) – Specification
- 3 – www.itsnet.com:~treelite:HL:BGH.html: Healing Light: Bovine Growth Hormone
- 4 – www.nalusda.gov:bic:BST:ndd:USE_OF_BOVINE_SOMATOTROPIN_IN_DAIRY_PRODUCTION.html: USE OF BOVINE SOMATOTROPIN IN DAIRY PRODUCTION
- 5 – users.aisp.net:wrabbit:humor:mcdonald.txt:
- 6 – www.labspec.co.za:l_meat.htm: LabSpec – Meat Allergens
- 7 – labspec.co.za:l_meat.htm: LabSpec – Meat Allergens
- 8 – ftp.cdrom.com:pub:obi:DEC:humor:Personnel-Office.in.Government:
- 9 – www.animalwelfare.com:farm:: Farm Animals
- 10 – www.wam.umd.edu:~markv:nmc.htm: NMC Workshop Bookmarks
- 11 – www.smithdairy.com:: All About Smith Dairy, The Dairy In The Country
- 12 – ss.niah.affrc.go.jp:bse:bse-a.html: Bovine Spongiform Encephalopathy (BSE-"Mad Cow Disease")
- 13 – www.livestockbreeders.com:: Livestock Breeders
- 14 – www.access.digex.net:~holycow:cgwprosres.htm: Prospect Technologies Personnel
- 15 – www.bright.net:~fwo:BSE:bse.html: Owenlea Farm Biosecurity/BSE Page
- 16 – www.mckinley.com:magellan:Reviews:Life_and_Style:Animals:Farm_Animals:index.magellan.html: Farm Animals
- 17 – www.pb.net:spc:mii:970746.HTM: MII News Article
- 18 – res.agr.ca:PUB:CDRN:portfoli:private:pierre:publ.html:
- 19 – www.ababeefalo.org:aba6.htm: Why Raise Beefalo

AltaVista

- 1 – www.urbancow.com:: The Urban Cow – Bovine Online – Virtual Cards
- 2 – www.urbancow.com:company.html: The Urban Cow – Bovine On-Line
- 3 – www.biotech.ist.unige.it:cldb:st219_40.html: Short description of cell lines. Strain/species: Bos taurus cow , bovine
- 4 – www.biotech.ist.unige.it:cldb:st220_40.html: Short description of cell lines. Strain/species: Freisian cow , bovine
- 5 – ss.niah.affrc.go.jp:bse:bse.html: Bovine Spongiform Encephalopathy (BSE-"Mad Cow Disease")
- 6 – http1.brunel.ac.uk:~hssrsdn:bse:article.htm: Bovine spongiform encephalopathy (BSE), or mad cow disease
- 7 – ss.niah.affrc.go.jp:bse:bse-for-s.html: Bovine Spongiform Encephalopathy (BSE-"Mad Cow Disease")
- 8 – ss.niah.affrc.go.jp:bse:bse-j.html: Bovine Spongiform Encephalopathy (BSE-"Mad Cow Disease")
- 9 – ss.niah.affrc.go.jp:bse:bse-u.html: Bovine Spongiform Encephalopathy (BSE-"Mad Cow Disease")
- 10 – ss.niah.affrc.go.jp:bse:bse-s.html: Bovine Spongiform Encephalopathy (BSE-"Mad Cow Disease")

The average number of bytes in the 29 files in cow was 10388.655273

Bookmarks

- 1 – www.prism.gatech.edu:~gt8572b:bookmarks.html: Maria Gullickson's Bookmarks
- 2 – www.movielink.com:: MovieLink | 777-FILM Online
- 3 – www.lycos.com:: Lycos: Your Personal Internet Guide
- 4 – www.emerson.emory.edu:services:latex:latex_toc.html: LaTeX help 1.1 – Table of Contents

The average number of bytes in the 4 bookmarks was 6936.250000

The total number of bytes downloaded was 333895

Figure 1: Index of Results from WebSnatcher

old data when retrieving new pages and creates an index to help users navigate new data. Users may optionally archive old data for future use.

Files are retrieved by requests sent to a server's HTTP port. For searches, a request is sent to the search engine's server, WebSnatcher parses the results to find matching pages, and then it retrieves these pages. Along with the main HTML file, WebSnatcher retrieves any images and frames contained within the file.

We have discussed several planned improvements for WebSnatcher, including a more robust and user-friendly interface to the profile. We also discussed incorporating an Anycasting scheme to allow users to specify a set of sites or servers, any of which could be used to satisfy a query; by monitoring past performance of different servers, WebSnatcher could guess which server will provide the best performance.

References

[1] AltaVista: Main Page. Web Site, 1997. http://www.altavista.digital.com.

[2] Samrat Bhattacharjee, Mostafa H. Ammar, Ellen W. Zegura, Viren Shah, and Zongming Fei. Generalized anycasting. In *Infocom '97*. IEEE, 1997. http://www.cc.gatech.edu/fac/Ellen.Zegura/papers/alas.ps.gz.

[3] Ann Chervenak. The Personal Terabyte Project. Web Site. http://www.cc.gatech.edu/fac/Ann.Chervenak/ptera/ptera.html.

[4] Fred Douglis, Antonia Haro, and Michael Rabinovich. HPP: HTML Macro-Preprocessing to Support Dynamic Document Caching. In *Symposium on Internet Technologies and Systems*. USENIX, December 1997. http://www.research.att.com/ douglis/papers/hpp/.

[5] Richard V. Dragan. Advice From the Web. *PC Magazine*, 9 September 1997. http://www.zdnet.com/pcmag/features/advice/_open.htm.

[6] Excite. Web Site, 1997. http://www.excite.com.

[7] Firefly. Web Site, 1996. http://www.firefly.net/.

[8] Terry A. Gray. How To Search The Web: A Guide To Search Tools. 1997. http://daphne.palomar.edu/TGSEARCH/.

[9] HotBot. Web Site, 1997. http://www.hotbot.com.

[10] The Informant - Common questions. FAQ, 1997. http://informant.dartmouth.edu/about.html.

[11] Infoseek. Web Site, 1998. http://www.infoseek.com.

[12] LikeMinds, Inc. Web Site, 1997. http://www.likeminds.com/.

[13] Tong Sau Loon and Vaduvur Bharghavan. Alleviating the Latency and Bandwidth Problems in WWW Browsing. 1997. http://timely.crhc.uiuc.edu/Papers/usits_v1.ps.

[14] Lycos Home Page. Web Site, 1998. http://www.lycos.com.

[15] Net Perceptions. Web Site, 1997. http://www.netperceptions.com/.

[16] Michael Pazzani, Jack Muramatsu, and Daniel Billsus. Syskill & Webert: Identifying interesting web sites. In *Spring Symposium*. AAAI, 1996. http://www.ics.uci.edu/~pazzani/RTF/AAAI.html.

[17] Marilyn Pedram. Introduction to Search Engines, September 1997. http://www.kcpl.lib.mo.us/search/srchengines.htm.

[18] *Wget Manual*, 27 May 1997. http://www.lns.cornell.edu/public/COMP/info/wget/wget_toc.html.

[19] WiseWire Corporation. Web Site, 1997. http://www.wisewire.com/.

[20] Yahoo! Web Site, 1998. http://www.yahoo.com.