Crawler-Friendly Web Servers

Onn Brandman, Junghoo Cho, Hector Garcia-Molina, Narayanan Shivakumar
Dept. of Computer Science
Stanford, CA 94305.
{*onn,cho,hector,shiva*}*@cs.stanford.edu*

Abstract

In this paper we study how to make web servers (e.g., Apache) more crawler friendly. Current web servers offer the same interface to crawlers and regular web surfers, even though crawlers and surfers have very different performance requirements. We evaluate simple and easy-to-incorporate modifications to web servers so that there are significant bandwidth savings. Specifically, we propose that web servers export meta-data archives decribing their content.

1 Introduction

A *web crawler* is a program that automatically downloads pages from the Web. A typical crawler starts with a seed set of pages (e.g., yahoo.com and aol.com). It then downloads these pages, extracts hyperlinks and crawls pages pointed to by these new hyperlinks. The crawler repeats this step until there are no more pages to crawl, or some resources (e.g., time or network bandwidth) are exhausted. The aforementioned method is referred to in this paper as *conventional crawling.*

In many cases it is important to keep the crawled pages "fresh" or up-to-date, for example, if the pages are used by a web search engine like AltaVista or Google. Thus, the crawler splits its resources in crawling new pages as well as checking if previously crawled pages have changed. The need to revisit pages generates even more demands on the crawler. To illustrate, the Web is currently estimated to have 800 million pages and still growing rapidly. If these pages have to be checked say once a week, the crawler needs to download more than 1300 pages a second.

A *web server* serves Hypertext Transfer Protocol (HTTP) requests (e.g., GET and POST) from web surfers and crawlers. Given a request, the server responds with either (1) static pages (e.g., a person's home page), or (2) dynamically generated pages (e.g., from a database in response to some user input).

Currently, a web server treats a crawler in the same fashion as it treats a web surfer. That is, the crawler has to request for a web page just like a surfer. This primitive interaction mode makes crawling more expensive than it need be, and gives rise to the following problems.

1. **Freshness:** Since web pages are modified at different frequencies, the crawler typically does not know when to revisit these pages. Often the crawler periodically polls the pages to check if they have changed. If a page has not changed since the last visit, the web server has wasted its resources. These resources could have been better spent servicing users or sending crawlers updates of pages that have actually changed.

2. **Incomplete data:** Recent estimates indicate that commercial web search engines index only 6 – 12% of web pages [12]. We believe the main reasons for such a small fraction of pages being indexed are:

   - Search engines have limited network bandwidth (the limit is based upon how much they have purchased from an ISP). If their current network bandwidth requirements are reduced (e.g., by not revisiting unchanged pages), we believe they would crawl a larger fraction of the Web.
   - Crawlers typically crawl pages that are hyperlinked from some previously crawled page. Crawlers may not reach some important page simply because the page is many links away from the seed URL.
   - Many crawlers do not go very "deep" into websites (here depth refers to the number of directories in a URL). This happens because crawlers try to avoid downloading infinite numbers of dynamically created pages, or because they try to avoid bombarding a single server with too many requests.

3. **Performance impact on web sites:** When crawlers periodically revisit pages at a web site to gain updates, the web server uses its network bandwidth to respond to these crawlers. Small businesses with low bandwidth connections to the Internet are in a quandary. While they would prefer to be indexed by the popular crawlers, they also have to pay their local Internet Service Provider (ISP) for the bandwidth. In addition, these sites have to expend CPU and memory resources servicing these crawler requests, when these resources could be better utilized servicing their customer hits. All the above resources are wasted if the pages that are being revisited do not change frequently.

4. **Different media:** Media-specific crawlers that index audio, video and images are gaining in popularity. However these crawlers usually do not know what media files are available at a web site. Hence they adopt the following approach. First, they crawl HTML pages like a regular web crawler. When a downloaded page has a hyperlink to a file with the appropriate media extension (e.g., .GIF, .JPG or .mp3), the media-specific crawler then downloads the file. In this case, the HTML pages are crawled and downloaded only to locate media files. Also media files at a web site that are not hyperlinked from some HTML page will not be found by the media-specific crawler.

We believe these problems can be alleviated if crawlers and search engines use a more appropriate protocol that helps efficiently identify what can and should be copied. For example, let us assume that the web server maintains a file that contains a list of URLs and their last-modified dates for all pages available from that web server. Then, the crawler may first download the URL-list file, identify which pages have been modified since the last crawl from the list file, and request only the pages that have changed. This may significantly reduce the number of requests the crawler makes to the web server, and thus the web server may save its valuable resources being wasted by servicing these requests.

The file of available URLs can also help crawlers reach dynamically generated pages at a site. For instance, suppose that a web vendor offers a search form, from which users can reach pages describing products of interest. A conventional crawler does not know how to fill out the form, so it cannot obtain any of these dynamic pages. However, if the vendor is interested in exporting the pages (so they can be indexed by a search engine), it can add appropriate URLs to the URL-list file. Each of these URLs will contain the necessary parameters (e.g., delimited with question marks), so a server CGI script can generate the page.

In this paper we explore how web sites can export meta-data (e.g., URL-list files) and we quantify the potential gains. Of course, exporting meta-data is not a new idea. For instance, Unix FTP servers often offer a file that includes meta-data describing each available file. The Harvest [8] proposal includes a distributed indexing scheme where web servers distribute meta-data.

Our goal here is simply adapt the meta-data export idea to web crawling and to build a *convincing* case for it. That is, it is relatively clear that exporting meta-data can save resources in some cases. But we want to check if the saved resources are *expected to be significant enough* to merit extending web access protocols. If the case is strong enough, we would hope to convince web server (e.g., Apache) implementers to add the necessary functionality, so that meta-data can be exported and crawlers can take advantage of it.

In Section 2, we present an overview of possible web server strategies to deal with crawlers. In Section 3 we describe how and at what cost web servers can export meta-data. In Section 4 we measure the efficiency of conventional crawling. In Sections 5 and 6 we evaluate the effects of meta-data availability. Section 7 addresses related work. Finally, Section 8 reports our conclusions.

2 Possible Web Server Strategies

Before proceeding, we briefly review options for improving the crawling process and explain why we prefer the meta-data option. There are at least two ways a server can assist a crawler:

1. **Help the crawler identify recently changed material.** Currently a crawler must request a page to discover if it has changed.

2. **Help the crawler discover pages without having to download and parse HTML pages.** The only way a crawler can find the location of pages is by downloading HTML pages and following links. Much of the downloaded HTML is just a means to discover some smaller set of desired pages.

As mentioned in the introduction, a web server may help the crawler by providing a list of all URLs and their meta-data. The crawler can use the meta-data to identify pages that have changed recently (i.e. since the last crawl). The meta-data also enables the crawler to discover what pages exist, including dynamic pages. This means the crawler could download all images without having to download any HTML pages. The crawler can then request each relevant page from the web server.

Another way the web server can accomplish goals 1 and 2 is to provide a new filtering interface, such as proposed in the DASL [10] standard. The crawler would specify a set of filter conditions (e.g., all pages that have changed after 1/10/2000) and the server then returns (using a single HTTP connection) all pages that meet the conditions. This strategy accomplishes both goals; recently changed material or certain types of pages can be amalgamated and sent to the crawler. This approach consumes less network bandwidth than the meta-data approach because the crawler does not have to download any meta-data or send a request for each page.

The server side filtering approach may be the most efficient, but it requires significant functionality to answer requests. The web server appears to be a database of pages, and must provide many of the functions a database system offers. We believe that such functionality is out of the scope of current web server implementations. The meta-data approach, on the other hand, on requires simple extensions to current web servers. In particular, the server only needs to periodically scan through the filesystem and store the collected meta-data into one or more files. For these reasons, in this paper we only focus on the meta-data approach.

3 More about Exporting URL Meta-Data

This section describes exactly what meta-data is exported, how it is accomplished, and at what cost.

3.1 What is the meta-data?

The exported meta-data should describe the entire website. The meta-data includes the size, last modified date, and path of each available page. These attributes are exported so that crawlers can:

- identify recently changed material
- find files matching certain media types
- know ahead of time how much bandwidth they will use before they download any pages

3.2 How big is meta-data?

Intuitively, one might reason that meta-data is small. Let's measure the size the meta-data entry for a typical web page. Assuming the file has reasonable path length of 30 characters, a size description of 5 characters (allowing maximum size of 99999 bytes), and an arbitrary last modified date, such an entry would look something like:
`‘‘05/Feb/1999:16:01:30 -0800 people/pics/line/colorbars.gif 99999’’`

To store this entry as an uncompressed ASCII file on most filesystems takes 64 bytes. Compression usually reduces the size of text files by a factor of three. This means a kilobyte fits 48 entries and 100KB can store meta-data for roughly 5000 pages. Most websites contain well under 5000 pages [15]. For these sites, all the meta-data can be described in a single 100KB file. 100KB is about the size of a large HTML page or a mid-sized image, so the cost of exporting meta-data is modest. In the next segment we visit the case of large websites.

3.3 Large websites

Some sites, such as `http://www.stanford.edu`, contain over 100,000 pages. These sites require over two megabytes to describe their content. In many cases a crawler does not require a meta-data file describing the entire site. In these cases a much smaller file could be used, resulting in bandwidth savings. For example, consider the case where a crawler is interested in HTML pages that have changed in the last week. If the server offered only a single file containing information about all of its pages, the crawler would download meta-data about too many URLs, resulting in wasted bandwidth.

To lower the overhead, web servers can first partition the meta-data into a set of compact archives. Interested crawlers then choose between the precomputed archives and download only what they need. The partitioning scheme should address the following issues:

- **Network overhead caused by giving crawlers too much information.** Crawlers should not download lots of meta-data that they do not need.
- **Too many files.** If crawlers need to request thousands of meta-data files then the web server will be burdened by such requests.
- **Server resources.** The partitioning process should consume reasonable amounts of CPU resources and disk space.

In this paper we do not study different partitioning schemes. Rather, we suggest single strategy and study the effects. Our partitioning scheme is as follows:

1. All page descriptions are first ordered by last modified date.
2. The sorted list is broken into segments of 2000 page descriptions each (our choice for the size of the segments is shortly explained).
3. Each segment is then broken into a set of smaller groups according to MIME type. Very small groups (e.g., 50 page descriptions) can be merged with other small groups into an "other" category.
4. Finally, each group of page descriptions is compressed and materialized into a file. The files are made available to crawlers in a standard place.
5. Steps 1-4 are repeated daily.

This partitioning scheme enables crawlers looking only for recently changed material and crawlers looking only for a specific MIME type to download sets of small, relevant archives rather than a single large archive. If we assume as before that the average page description is 64 bytes long and take the maximum number of page descriptions per file (2000) into account, the largest compressed meta-data file will be about 42KB. Assuming that Step 3 splits the segments into five different MIME groups, a website with 100,000 pages will have about 250 archives. We feel this is a reasonable compromise between avoiding large files and avoiding too many files.

Crawlers looking for recently changed material will never download more than 42KB of irrelevant meta-data. For example, say a crawler needs all pages that have changed on or after 1/10/2000. A web server employing our partitioning strategy might have an archive of pages last modified from 1/10/2000 to 3/2/2000. This archive is likely to contain mostly irrelevant page descriptions so the crawler will download up to 42KB of extra meta-data.

Media specific crawlers can often download exactly what they want without downloading any other media type. If they seek a type that has been grouped into the "other" category, then they will have to filter out some URLs from the meta-data they receive. However, as is shown in Section

4, the "other" category typically makes up about 15% of total pages. Thus the amount of irrelevant meta-data such crawlers download will usually be no larger than 15% of the total meta-data.

Since each page description appears in exactly one archive, the disk space required for our scheme is as low as possible. The main CPU cost comes from compressing the archives. In some simple experiments, we determined that compressing a 42KB file on a 400 MHz PC running Linux consumes 0.010 CPU seconds, which is a modest cost. Depending on the number of served pages, scanning though the file system can incur many I/O's. To offset the cost of compression and I/O's the web server may spread the task over an entire day.

Regardless of the partitioning scheme chosen by the server, crawlers need to be able to know about the contents of the archives so that only the appropriate archives are downloaded and bandwidth is conserved. This requires a standard so that the web server can effectively communicate the contents of each file to the crawlers. We do not fully address this issue in this paper. However, one possible strategy is to use a file naming scheme. For example, `1.4.2000-1.22.2000_HTML.gz` could represent a compressed archive describing HTML pages modified between 1/4/2000 and 1/22/2000. The robots.txt file (which already exists on most web servers) could list the names of available archives and where they are located. In this paper we assume that crawlers are able to download the minimum number of archives to suit their needs.

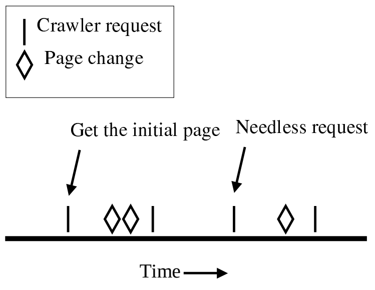
3.4 Does exporting meta-data incur significant network overhead?

We can make some assertions about the network overhead resulting from our partitioning scheme. Crawlers downloading a site for the first time might download all the available archives. Using our previous assumption that the size of an average page description is 64 bytes, the size of a compressed page description is 22 bytes. On our web servers, we measured the the average size of a web page to be 13KB. This means the size ratio of meta-data to data is about 1:600, or 0.16% overhead. Thus, the overhead incurred by downloading the complete archives is insignificant compared to the cost of downloading all content from a website.

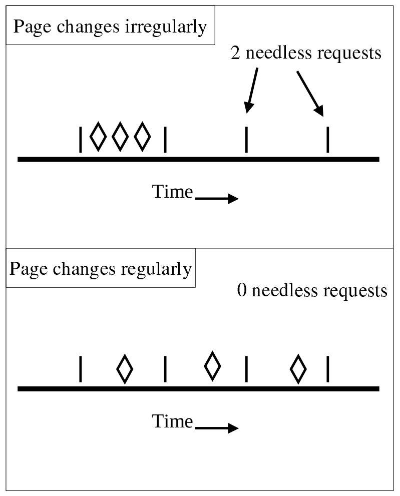
When these crawlers return to update their indexes, they will only download archives of recently changed material. If many things have changed a large amount of meta-data will be downloaded. If few pages have changed then the amount of meta-data downloaded will be small. As long as the amount of meta-data transfered is proportional to the amount of data transfered, the relative overhead is still around 0.16%.

4 The Efficiency of Conventional Crawling

In order to understand the impact of meta-data availability, we evaluate the efficiency of conventional crawling. We first measure the wasted resources that result from a conventional crawler's efforts to maintain a repository. Here maintainence refers to updating the repository over time to reflect changes in the Web. Finally we evaluate the efficiency with which a conventional crawler can build a media specific repository (e.g., an image repository).



(a)



(b)

Figure 1: Needless requests

4.1 Maintaining a repository

Under the conventional crawling scheme, a crawler must make an HTTP request to find out if a page has changed. If the page has not changed then the HTTP request was needless. Figure 1(a) illustrates how needless downloads occur.

Say a page changes C times in a fixed time period while a crawler requests the page R times at regular intervals. Figure 1(b) shows how the distribution of page changes has an effect on the number needless requests. We can calculate a lower bound for the number of needless requests for a single page using the formula:

$$MinNeedlessReq(C, R) = \begin{cases} R - (C+1) & \text{if } R > C - 1 \\ 0 & \text{otherwise} \end{cases}$$

We increment C by one to account for gathering the initial version of the page. For example, consider a page that changes five times over a month and a crawler that downloads the page ten times for that month. The crawler's first request is not needless because the crawler has no copy of the document. However, of the subsequent nine requests, at least four are needless.

Note that this lower bound corresponds to page changes and crawler requests being as interleaved as possible. This implies a distribution of page changes where the number of page changes that occur between two consecutive crawler requests is as low as possible.

To assess how many pages are needlessly requested and/or downloaded by conventional crawlers, we need to know how often web pages change. To obtain the change rates of web pages, we traced

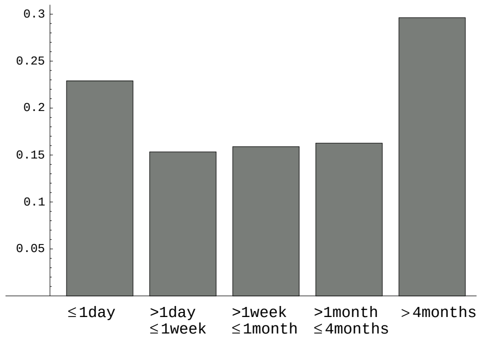


Figure 2: Change frequency distribution

the change history of 760,000 web pages from the 270 "most-linked" sites for over 4 months. That is, we repeatedly visited those 760,000 pages every day and checked whether the pages had changed or not. This gave us the change frequency for each page. For example, if a page changed 4 times in the 4 months, we can reasonably say that the page changes once every month on average. Figure 2 shows the statistics we collected. In the graph, the horizontal axis shows the average change interval and the vertical axis shows the fraction of pages that changed at that change interval. For example, from the first bar of Figure 2, we can see that around 23% of pages changed at least once every day during our experiment.

We can use our collected data to assess how many pages are needlessly requested by a conventional crawler. For example, when the crawler revisits pages once every month the pages that have changed more than once every month (the first three bars of Figure 2) will have changed whenever the crawler visited the page, so there are no needless downloads for these pages. However, some of the pages corresponding to the fourth and the fifth bars may have not changed between visits and may have been needlessly requested. We calculated the minimum number of needless requests occurring when crawling all 760,000 pages for 100 days. For each of the 760,000 pages, we use the measured number of changes C and compute the lower bound $MinNeedlessRequests$ given some number of crawler visits R (the same for all pages). We add all the lower bounds to obtain the least total needless downloads. Finally, we divide this total by the computed numbr of visited pages (R times 760,000), to obtain the fraction of useless downloads, shown in Figure 3.

The horizontal axis of Figure 3 represents the number of times each page was visited by the crawler during the 100 day period and the vertical axis shows the fraction of needless downloads. We can see that the crawlers download more and more needless pages as the they visit pages more often. For instance, when the crawler visits all pages once every day, around 77% of downloads are needless. This is because 23% of the pages change at least once a day (Figure 2, first bar). We can also see that the fraction of needless downloads increases very rapidly in the beginning and then levels off. This rapid increase is because a large number of pages change very slowly. From the fifth bar of Figure 2, we can see more than 30% of pages changes less than once every 4 months. These pages are often requested needlessly, even when the crawler requests pages a relatively low number of times.

Needless requests that occur in the crawler's maintainence process need not result in needless downloads. The crawler can use HTTPs `If-Modified-Since` feature to avoid download loading

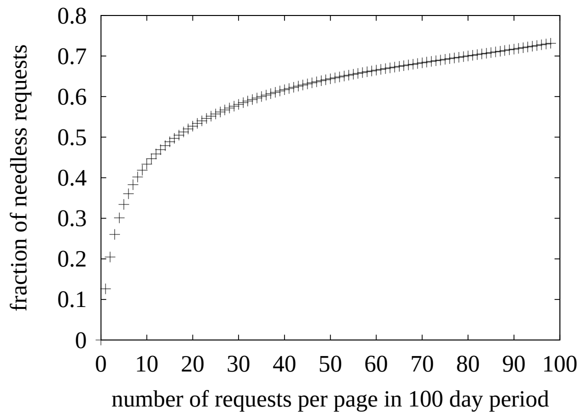


Figure 3: Needless requests

a file as a result of a needless request. This feature causes the web server to only transfer a file if it has change since some date (e.g., the date of the last crawl). However, many crawlers ignore this feature and many web servers do not export this functionality. To estimate whether or not crawlers were using HTTP's `If-Modified-Since` feature, we analyzed eight [1] Stanford server access logs. Since most web pages change slowly, crawlers using the `If-Modified-Since` feature should sometimes receive a response code of 304 `Not-Modified` when requesting a document. We tracked IPs that downloaded the Robots.txt file to identify the IP addresses of crawlers. Assuming that these were crawlers, only 13% of crawlers got even a single server response of 304 `Not-Modified`. This implies that 87% of crawlers do not check time-stamps before downloading a page. For such crawlers, needless requests result in needlessly downloaded pages, wasting even more web server (and crawler) resources.

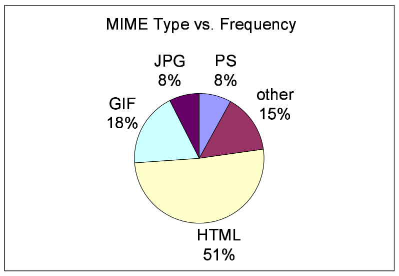
4.2 Building a Media Specific Repository

Another set of needless requests occurs in the page discovery process. When a conventional crawler seeks to download a certain MIME type, it is forced to first download all available HTML pages. This is because if even a single HTML page is skipped the crawler takes the risk of not discovering media files linked from that page.

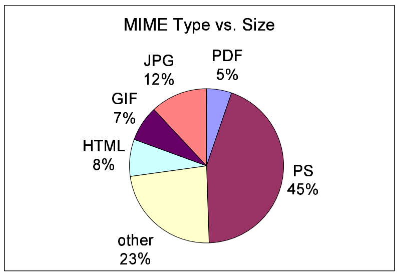
To find out how much overhead is caused by downloading HTML, we measured the MIME type distribution for eight Stanford websites. We examined web server access logs files to approximate what pages are available at each site (we did not have access to the files, only to the access logs). This assumes that every available page was accessed at least once (by crawler or human user) and thus appeared in the logs. Figure 4(a) shows the relative number of pages in each MIME type. MIME types that represent less than 5% of the pages are grouped into the *other* category.

Clearly the number of HTML pages overshadows all other MIME types. Since all HTML pages must be downloaded to gather the other MIME types, many unnecessary hits occur. For example, say a conventional crawler wants all images (JPEG and GIF files) and downloads all HTML pages

[1]www-db.stanford.edu, www-flash.stanford.edu, www-diglib.stanford.edu, theory.stanford.edu, csl.stanford.edu, robotics.stanford.edu, graphics.stanford.edu, cs.stanford.edu



(a)



(b)

Figure 4: MIME type distributions

to discover them. In this case 66% of the requests are for HTML rather than images.

We also plotted the amount of data in each MIME type. This is useful to see how much bandwidth is wasted when the conventional crawler searches for a certain MIME type. Figure 4(b) shows the relative amounts of bytes in each MIME type. Again, MIME types accounting for less than 5% of the total size are grouped into the other category. Consider our previous example of a crawler downloading images. Since the crawler must download all available HTML files in addition to image files, 30% of the used bandwidth is wasted.

5 The Impact of Meta-Data on web Server Resource Consumption

The inefficiencies of conventional crawling revealed in the previous section can be avoided if servers export meta-data. During the repository maintainence process, a crawler would never needlessly request a page that has not changed because the meta-data reveals what pages have changed. When building a media specific repository, a crawler would never need to download any HTML, just the desired media. In this section we examine the impact a streamlined crawling process will have on web servers.

Using web server access logs, we approximated crawler traffic for eight Stanford web servers. Figure 5(a) shows the amount of crawler traffic vs. the total traffic for our web servers. Each point represents one of our web servers. As can be seen in the figure, crawler traffic increases with total traffic. Needless crawler downloads account for many requests at popular websites and fewer requests at less popular sites.

While less popular sites have a low absolute amount of crawler traffic, they have a high relative amount of crawler traffic. Figure 5(b) shows the percentage of traffic due to crawlers vs. total traffic for our web servers. Each point represents one web server. A line is superimposed on the figure to illustrate the linear behavior of the distribution. The percentage of traffic due to crawlers decreases as the total traffic increases with a slope of $-.47$ % points/1000 hits. For example, at

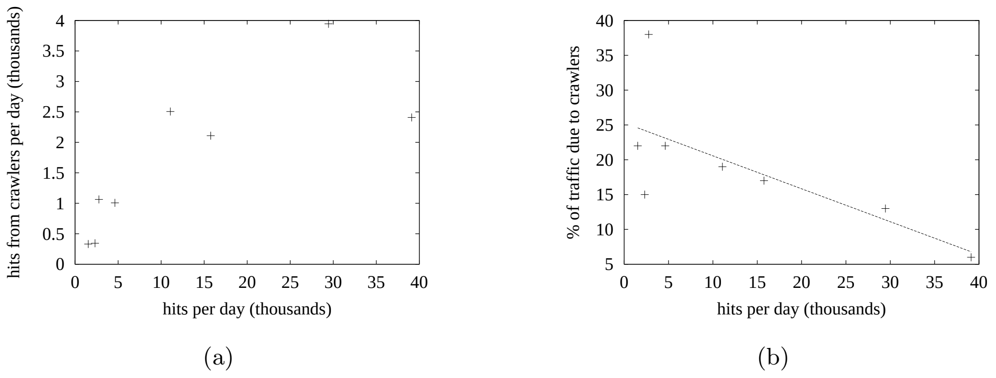


Figure 5: crawler traffic characterization

10,000 hits per day, crawler overhead is about 18%. If we double daily traffic to 20,000 hits a day, the overhead its $18 - 4.7 = 13.3\%$. Eliminating needless crawler requests will have an impact on less popular websites because a large percentage of their hits come from crawlers.

6 The Effects of Meta-Data Availability on Freshness

A crawled page is fresh when the crawler's copy matches the web server's copy. It is in the interest of web servers to support an efficient crawler refresh strategy because:

1. Search engines can then work with a more accurate representation of the websites, directing users to a website only when appropriate.
2. An efficient refresh process will have an effect on the size of the repository. Most search engines index about 6%-12% of the web [12], largely due to crawlers dividing their resources between exploration and updating their pages. By reducing the cost of maintaining a repository, crawlers can keep larger repositories and websites are more likely to be included.
3. The refresh process consumes web server resources. If a crawler seeks to keep a very fresh repository then many web severs' resources might be wasted.

Keeping an archive fresh consumes bandwidth. For a conventional crawler that uses the same request frequency for all pages, limited bandwidth results in a limited request frequency. That is, reducing the crawler's available bandwidth forces it to visit pages less often. If a page changes often but is seldomly downloaded by the crawler then the page is unlikely to be fresh in the crawler's repository. For a meta-data based refresh method, a bandwidth limit limits the amount of meta-data and changed data that can be downloaded. If a crawler knows (by looking at meta-data) that a page has changed then the crawler may be unable to download the page for lack of available bandwidth.

We compared the efficiency of a conventional crawler to a meta-data based crawler by measuring repository freshness (the fraction of fresh pages in the repository) as a function of available bandwidth. We ran simulated crawls of 10000 pages from four sites over a time period of 100 days. Simulated crawls occurred at 100 different levels of available bandwidth for both types of crawlers (conventional and meta-data enabled). At the beginning of each simulated crawl, each crawler had a fresh repository of pages. The fraction of pages fresh the *end* of the 100 day crawling period was recorded. The change frequencies for each of the 10000 pages in this simulation came from the experimental data described in Section 4. We made the following assumptions:

- All pages were of equal size. This means the bandwidth limit is equivalent to a cap on the number of pages that each crawler can download in one day.
- Each day, the meta-data enabled crawler downloaded two 42KB meta-data archives from each site. This consumed the same amount of bandwidth as downloading 8 regular pages (we again make the assumption that the average page size is 13KB).
- The conventional crawler's needless requests result in needless downloads. As stated in Section 4, this is usually the case.
- The pages change at regular intervals. This was done to simplify the involved calculations for our simulation.
- Web servers publish meta-data about their pages every day.
- Pages do not change more than once a day. That is, a page downloaded on a given day is guaranteed to be fresh until the following day.

Given a daily page limit L, the conventional crawler downloaded pages during the 100 day period using a round-robin download schedule as follows:

1. All pages to be crawled were first ordered in some arbitrary fashion.
2. Every day the conventional crawler downloaded the next L pages according to the predefined order. When the end of the list was reached the crawler started at the beginning.

Given a daily page limit L, the meta-data based crawler downloaded pages as follows:

1. All pages to be crawled were first ordered in some arbitrary fashion.
2. Every day the crawler downloaded meta-data from each site. The meta-data is then examined so that changed pages can be identified.
3. Every day the crawler downloaded the next L *changed* pages according to the predefined order. When the end of the list was reached the crawler started at the beginning. If less than L pages changed then the crawler stopped downloading pages that day.

Figure 6(a) shows the fraction of fresh pages in the each crawler's repository as a function of available bandwidth. The maximum bandwidth value of 1 corresponds to having enough available bandwidth to download the entire repository in one day. That is, the daily page limit L is equal to the number of pages in the repository (this is unattainable in most "real world" situations).

Trajectories are plotted for both conventional crawling and meta-data based crawling. As can be seen in the figure, meta-data availability allows a fresher repository to be maintained at lower bandwidth. The meta-data enabled crawler is able to maintain a perfectly fresh repository at about a fifth of the bandwidth required by the conventional crawler.

Using meta-data allows an additional fraction of the repository to remain fresh. For a given amount of bandwidth, we can discover the additional gains by computing the difference between the associated repository freshness of the crawlers and dividing by the freshness of the conventional crawler. Figure 6(b) illustrates how the gains change with bandwidth. At low bandwidth using meta-data increases repository freshness by up to 25%. When the bandwidth reaches a value of .22 (this means 22% of the available pages can be downloaded each day) the benefits of using meta-data begin to decrease. This suggests that using meta-data is most useful to crawlers with limited bandwidth. Of course, given the growth and size of the Web, we expect most crawlers to be bandwidth limited, in the range 0 to 0.2 bandwidth.

If the crawler desires to keep a very fresh repository, using meta-data reduces the bandwidth required. Figure 6(c) shows the percentage of bandwidth saved by using meta-data in the crawling process as a function of desired repository freshness. For example, if the crawler desires a repository that contains 80% fresh pages, it can save 62% of the bandwidth costs.

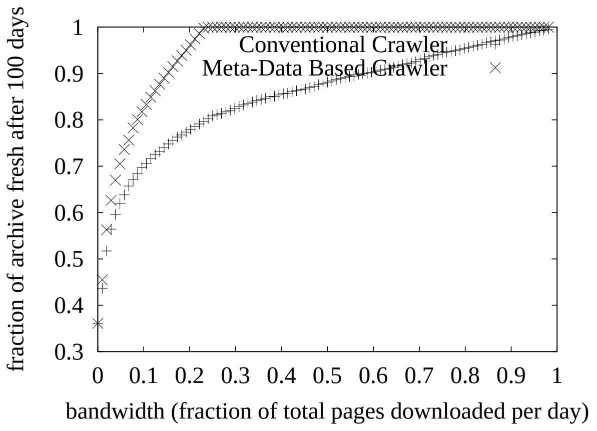
7 Related Work

There have been a variety of proposals to help retrieve information over distributed data by exporting meta-data. However, most of the proposals require web servers to analyze and summarize the *content* of their data. For example, under the Harvest [8] proposal, each website maintains an *inverted index* for its pages. The STARTS [7] proposal enables searching over a set of information sources. Meta-data from websites is analyzed to choose the best sources to evaluate a query. Since these proposals are complex and demand extra work from web servers, they have not been widely deployed. Our work, on the other hand, only requires web servers to list the location and the last-modified date of their data. This is both less computationally intensive and less complex, making it more plausible to deploy.

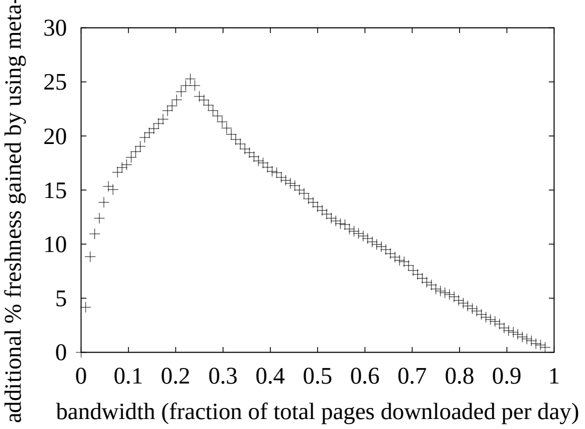
Some standards focus on the representation of meta-data. RDF [2] integrates a variety of web-based meta-data activities, including sitemaps and search engine data collection, digital library collections, using XML [9] as an interchange and storage standard. The Dublin Core [1] provides a common core of semantics for resource description. Document attributes such as Name, Identifier, Version, Language are precisely defined. Dublin Core is often implemented via RDF. We believe these standards can be used to export meta-data as we have proposed.

Many standards exist to enable client to server querying. Z39.50 [3] is an application layer, session oriented protocol for search and retrieval. DASL[10] is a standard for server side searching. The client can formulate a query and have the server identify the resources that fit the query. With these standards, crawlers can issue queries to identify relevant data. However, as discussed in section 2, we prefer a meta-data based approach because it demands less from web servers.

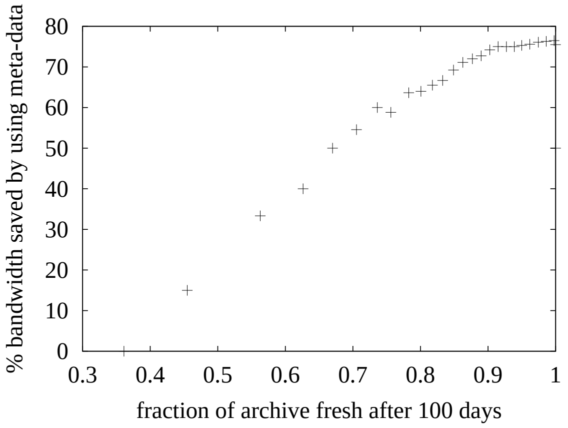
References [6, 5] study how a crawler can increase the "freshness" of its collection by visiting pages at different frequencies, based on how often the pages change. Reference [4] studies how a crawler can *estimate page change frequencies*, by accessing the pages repeatedly. We believe



(a)



(b)



(c)

Figure 6: Freshness comparisons

crawlers can estimate the change frequency better and eliminate needless downloads if our proposal is adopted. For example, reference [4] shows that a crawler can estimate change frequency much more accurately, when the *last-modified date* is known. In our proposal, the last-modified dates for all pages are available in a meta-data file, which can be easily accessed by crawlers.

8 Conclusion

We propose that web servers export meta-data describing their pages so that crawlers can efficiently create and maintain large, "fresh" repositories. This meta-data includes the last modified date and size for each available file.

To evaluate the effects of our proposals, we gathered evolution data for 760,000 pages and several web server access logs. We ran simulated crawls using both the typical crawling strategy and a meta-data enabled crawling strategy. From our data and experiments we conclude that significant web server resources (e.g., bandwidth) can be saved if servers export meta-data.

References

[1] The dublin core metadata initiative. http://purl.oclc.org/dc/, 1998.

[2] *Resource Description Framework (RDF) Model and Syntax Specification*, 1998. Available at http://www.w3.org/TR/WD-rdf-syntax/.

[3] ANSI/NISO. *Information Retrieval: Application Service Definition and Protocol Specification*, April 1995. Available at http://lcweb.loc.gov/z3950/agency/document.html.

[4] J. Cho and H. Garcia-Molina. Estimating frequency of change. Technical report, Stanford University, 2000.

[5] J. Cho and H. Garcia-Molina. Synchronizing a database to improve freshness. In *Proceedings of the 2000 ACM SIGMOD*, 2000.

[6] E. Coffman, Jr., Z. Liu, and R. R. Weber. Optimal robot scheduling for web search engines. Technical report, INRIA, 1997.

[7] L. Gravano, C.-C. K. Chang, H. García-Molina, and A. Paepcke. STARTS: Stanford protocol proposal for Internet retrieval and search. Technical Report SIDL-WP-1996-0043, Stanford University, Aug. 1996. Accessible at `http://www-diglib.stanford.edu/cgi-bin/WP/get/-SIDL-WP-1996-0043`.

[8] D. R. Hardy, M. F. Schwartz, and D. Wessels. Harvest user's manual, Jan. 1996. Accessible at `http://harvest.transarc.com/afs/transarc.com/public/trg/Harvest/user-manual`.

[9] E. Miller. An introduction to the resource description framework. *D-Lib Magazine*, May 1998. http://www.dlib.org/dlib/may98/miller/05miller.html.

[10] S. Reddy, D. Lowry, S. Reddy, R. Henderson, J. Davis, and A. Babich. Dav searching & locating, internet draft. Technical Report http://www.webdav.org/dasl/protocol/draft-dasl-protocol-00.html, IETF, June 1999.

[11] Venkata N. Padmanabhan, Jefferey C. Mogul. *Improving HTTP Latency.* The Second International WWW Conference, Chicago, IL, USA, Oct 1994
http://www.cs.berkeley.edu/˜padmanab/papers/www_fall94.ps

[12] Steve Laurence, C. Lee Giles. *Accessibility of Information on the Web.* Nature, Vol. 400, pg. 170 July 8, 1999
http://www.nature.com/server-java/Propub/nature/400107A0.pdf

[13] Van Jacobson. *Congestion Avoidance and Control.* In Proc. SIGCOMM '88 Symposium on Communications Architectures and Protocols, pages 314-329. Stanford, CA, August, 1998

[14] Jeffery C. Mogul. *The Case for Persisten-Connection HTTP.* Digital WRL Research Report 95/9, May 1995
http://www.research.digital.com/wrl/publications/abstracts/95.4.html

[15] Inktomi Corporation *Inktomi Webmap*
http://www.inktomi.com/webmap

[16] Paul Barford, Mark Crovella. *Generating Reprasentative Web Workloads for Network and Server Performance Evaluation.* SIGMETRICS June, 1998
http://www.cs.wpi.edu/ sigmet98/barford.ps

[17] Jeffery C. Mogul, Fred Douglis, Anja Feldmann, Balachander Krishnamurthy. *Potential Benefits of Delta Encoding and Data Compression for HTTP.* Compaq Western Research Laboratory, Research Report 97/4, July 1997
http://www.research.digital.com/wrl/techreports/abstracts/97.4.html