Web site archiving - an approach to recording every materially different response produced by a website

Kent Fitch, Project Computing Pty Ltd. Kent.Fitch@ProjectComputing.com

Abstract

With the critical role played by web servers in corporate communication and the recognition that information published on a web site has the same legal status as its paper equivalent, knowing exactly what has been delivered to viewers of a web site is as much a necessity as keeping file copies of official paper correspondence.

However, whilst traditional records management, change control and versioning systems potentially address the problem of tracking updates to content, in practice, web responses are increasingly being generated dynamically: pages are constructed on the fly from a combination of sources including databases, feeds, script output and static content using dynamically selected templates, stylesheets and output filters and often with per-user "personalisation". Furthermore, the content types being generated are steadily expanding from HTML text and images into audio, video and applications.

Under such circumstances, being able to state with confidence exactly what a site looked like at a given date and exactly what responses have been generated and how and when those responses changed becomes extremely problematic.

This paper discusses an approach to capturing and archiving all materially distinct responses produced by a web site, regardless of their content type and how they are produced. This approach does not remove the need for traditional records management practices but rather augments them by archiving the end results of changes to content and content generation systems. It also discusses the applicability of this approach to the capturing of web sites by harvesters.

Keywords: web site archiving, web site harvesting, web server filters, managing electronic records

Introduction

Web sites are now widely used by business and government to publish information. Increasingly, these web sites are becoming the major or only means of disseminating information from the publisher to the public and other organisations.

Internal information which formerly would have been circulated as manuals or memos is also increasingly being published exclusively on an intranet web site.

Business and Governments publish on the web because of convenience, speed, flexibility and cost. Government initiatives such as Australia's "Government Online" (HREF2) and the US's "Government Paperwork Elimination Act" (HREF3, HREF4) require government agencies to publish online.

Because electronically published documents are increasingly taking over the roles of paper publications, organisations have a clear need to record exactly what has been published and how it was presented to users of their websites.

A document on a web site is effectively republished (or 'performed') each time it is viewed. This subtle point not only cuts to the heart of the copyright issues associated with digitisation projects and digital libraries (HREF5) but also to the general problem of web archiving. On each publication event the response to the request is generated anew. A large and often complex software system uses algorithms to convert one or more sources
of input data into a response. Changes to the input data, algorithms or often even subtle
details of the request such as the apparent address of the client or the characteristics of
their browser can alter the response.

The Netcraft Web Server survey for December 2002 (HREF1) reported that the percentage
of IP addresses with a home page generated by script increased from 16% to 23%
between January and December 2002:

"The number of active sites has risen by around 17% over the last year,
indicating that the conventional web is still expanding at a respectable rate, and
the number of SSL sites is up by a roughly equivalent 14%. But most notably
the number of sites making some use of scripting languages on the front page
has increased by over half. ASP and PHP, which are by far the most widely
used scripting languages, have each seen significant increases in deployment on
the internet, as businesses constructed more sophisticated sites, upgrading
initial brochureware efforts."

There are several approaches to tackling the problem of recording what has been
published:

1. record every change to the generation software, the environment in which it runs and
the inputs it uses to generate the response
2. use a web spider to take regular "snapshots" of the website
3. make regular backup copies of the website
4. record every materially different response produced by the website

The contention of this paper is that the fourth approach produces a more complete and
faithful record than the alternatives and does so with significantly less cost and more
convenience.

Motivations for recording what has been published

There are several reasons for an organisation to take a great interest in efficiently and
effectively recording the often transient contents of their web sites, including:

1. Legal

Information provided on a website is now generally accepted to have the same legal
standing as its paper equivalent. Hence, an organisation must be prepared for a legal
argument based on what was or was not published by them on their website. It is
widely held that it is not just the text but it is the context and website "experience"
which is likely to be examined in a legal investigation of what information was
provided or whether a contractual agreement has been entered into (HREF4, HREF8,
HREF16, HREF17). In these circumstances, it is vital that an organisation is able to
credibly assert what has been published and how it was presented.

2. Community expectations, reputation and commercial advantage

Consumers and the community in general prefer to deal with ethical, reputable and
well-run organisations. It is important for an organisation to be perceived as standing
by their products and the information they've provided. In the paper based world,
consumers have the advantage of the immutable physical medium of paper. In the
transient electronic world, organisations offering an equivalent by way of a notarised
or otherwise reliable "archive" will attract discriminating consumers and may be able
to attach a price premium to their services.

In an increasingly competitive global marketplace, reputation is a key factor for
organisations wishing to differentiate themselves as trusted providers of information,
services and products. Hence, archiving practices that enable a complete publication record to be faithfully maintained and accessed should be seen as commercial assets, not just legal necessities.

3. Providence

There are many projects attempting to create archives of web sites of national significance ([HREF9](#), [HREF10](#), [HREF11](#), [HREF12](#), [HREF15](#)). Many organisations also seek to create internal archives as "company records".

Techniques for recording what has been published

As summarised above, there are several approaches to recording what has been published on a web site with a view to allowing the site to be reconstructed as it was at a particular time in the past. This section discusses the pros and cons of each approach.

A "perfect" approach would probably be a system in which everything ever presented in the browser of every visitor to the web site was recorded exactly as seen and was independently notarised and archived. Although such a system is impractical (because it would require screen capture on every visitor's machine and the archiving of those images) it provides a useful benchmark against which to measure alternatives.

The criteria used below for assessing the suitability of the various approaches are as follows:

- **Coverage.** Are all publication "events" covered? Can dynamically generated content be recorded? What about content originating from syndicated feeds, database transactions, real-time information (such as stock prices, weather etc)? How easy is it to accidentally or deliberately subvert the system? Can "non-material" changes be ignored by the system?

- **Robustness.** Does the system pass the "keep it simple" test? Can it be easily understood? Can it adapt as content generation and publication systems change?

- **Cost.** Is the approach economically feasible? Does it scale over time and multiple web sites?

- **Re-creation.** Can the web-site be effectively re-created at a nominated "point in time"? Does it faithfully reproduce exactly what was delivered to users of the web site? Can the system be used to spot trends and answer queries about content change activity?

Using the criteria, the following approaches are examined:

1. **Recording all changes to inputs and algorithms**

   **Process:**
   
   Systems are provided which track all changes made "on the input side" to all data and processes which effect the content of responses from a web site.

   **Pros:**
   
   i. Cost - often comes "for free".

   Change management/versioning controls are often built in to Content Management Systems (CMS) and hence come "for free". Such systems are required anyway for other business reasons, such as workflow, accountability and recovery. Developers frequently use versioning systems for program code
for the same reasons.

ii. Cost - data volumes are smaller.

Versioning systems frequently store just the differences or the "deltas" between versions. The differences are often small and hence require much less storage than complete copies of each version.

Cons:

i. Coverage - completeness

How can you be sure all changes will be recorded? How can you be even sure that you know of all the systems which impact the generation of web pages?

It is human nature to be "goal" directed rather than "task" directed. If something needs to be changed, the "goal" of changing it often overrides the proscribed "tasks" for changing it. Hence, changes are frequently made outside of mandated control systems either accidentally, or just to get the job done as efficiently as possible, or occasionally with malicious intent. As the number of separate systems making up the chain of contributors to content on a web site increases, the number of people and administrative areas involved increases and the risk of only partial coverage increases making later reconstruction of the web site impossible.

ii. Coverage - diversity of changeable components

Whereas a content management system may aim to track updates to most content and even sometimes templates and scripts, the scope of components required to be recorded is much larger, and includes operating systems, database systems and other software (and their patches), data (including transactional data) and access permissions.

Devising archiving/version control systems for some components is extremely difficult and expensive.

iii. Re-creation of the web site

Re-creating the web site requires reinstating the entire system as it was at the desired point in time and re-issuing the requests to the system originally made.

This is a hard task for stand-alone systems, where the catalogue of software required may be extensive (from the Operating System, as patched, up), but where external systems are used for input data or control data (such as authorisations), the job may be impractical. Where transactional databases are involved, the task of rolling back the state of the system to the point of the transaction may be impossible.

Furthermore, because the response seen by users is sometimes tailored based on client-provided information (cookies, browser user agent etc), client software and state information may also need to be reinstated. Effects of network components between the server and client (such as proxies and caches) may need to be considered.

2. **Use a spider to take regular snapshots**

   **Process:**

   Crawl the web site using a spider. Archive the results.
Pros:

i. Robustness - simple

Effective and simple spiders are freely available.

ii. Re-creation - faithful

HTTP responses are archived, not "raw" data. Hence the archive does not need the original execution environment to be recreated to allow the point-in-time view.

Cons:

i. Coverage - address-space incomplete

Spiders can only generate the HTTP requests they find by a recursive parsing and retrieval of page rooted from a set of one or more starting points. They cannot crawl parts of the web site where the requests are generated by, for example, HTML forms or client-side Javascript. So, much of the dynamic content of the site (eg, a search response) cannot be captured. Similarly, the responses to transactional systems cannot be captured. Variations based on some client configuration can be captured but only by rerunning the spider with a defined set of client configurations. Variations based on other client state (such as cookies) cannot be captured.

ii. Coverage - temporally incomplete

Changes made between spidering passes are not recorded.

iii. Cost - volume

A complete crawl of a large web site (and associated sites) at a reasonable frequency to minimize the risks of losing changes made between spidering passes may generate an extremely large volume of data.

iv. Re-creation of the web site

Because many dynamically generated responses can not be captured, a full recreation of the web site at a point in time is impossible.

As with the previous approach, the effect on responses of client user agent, cookies etc cannot be represented.

3. **Take regular backups of the website**

   Process:

   Create a backup of the website and all associated components. Archive the results.

   Pros:

   i. Cost - often a free byproduct of system operation

   Organisations typically already backup their computer systems to allow recovery from hardware and software failures, accidents and disasters.

   ii. Coverage - may be complete
System backups typically back up everything - from system software to user data (however, see "con" below).

Cons:

i. Coverage - doesn't address external content providers

Some of the data or processes involved in the generation of content may be external to the web server, eg, an authentication database running on a separate machine, a transaction database, a syndicated feed, etc. These systems may be backed up on separate cycles, or may not be effectively recoverable for the purposes of web site re-creation.

ii. Coverage - temporally incomplete

Changes made between backups are not recorded.

iii. Cost - volume

A complete backup of a large web system (and associated systems) at a reasonable frequency to minimize the risks of losing changes made between backups may be very large.

iv. Re-creation of the web site

Re-creation requires establishment of a complete system to a state where it can run the restored software. That is, hardware capable of running the system as backed-up must be available and operational. The continual maintenance of such hardware or the liability of its subsequent purchase becomes a cost of this approach.

Other systems (such as authorisation or transactional databases) which provide input to the response generation process must also be restored to the desired point-in-time, a process which is typically extremely expensive and risky.

As with the previous approaches, the effect on responses of client user agent, cookies etc cannot be represented.

4. **Recording all materially different responses**

Process:

Inspect all responses generated by the web server. Archive those which have not been seen before.

Pros:

i. Coverage - independent of content type, content change control and method of response generation

Not subvertable by clandestine updates.

ii. Coverage - address space and temporally complete

Every response is inspected. Response variations based on client state are recorded.

iii. Re-creation - faithful

HTTP responses are archived, not "raw" data. Hence the archive does not need
the original execution environment to be recreated to allow the point-in-time view.

iv. Cost - volume

Although many busy web servers generate gigabytes of responses per day, the volume of unique responses is often very low - frequently in just the megabytes or tens of megabytes per day as discussed in the next section.

Cons:

i. Cost - overhead

With busy webservers delivering hundreds of thousands of pages per day, the determination of unique responses must be performed more-or-less in real time but with minimal impact on the performance of the web server.

ii. Robustness - critical path

The software performing the determination of uniqueness needs to "hook" into the path between the client and the server. Failure of the hook results in either uncaptured (lost) responses or, potentially, failure of the server. (A flip-side of this characteristic is that if a critical piece of the archiving software fails, the web site also fails. This approach is common in transactional databases which fail if the database logging systems fail.)

iii. Coverage - non material changes

Responses to the same request may be different, but not materially different. For example, a home page which greets the user with the current date and time or weather forecast may not be materially different in any other respect (unless the web server provides a time or weather service!). Many sites make extensive use of such personalisation, sometimes on every page. Archiving responses that are not materially different creates unnecessary volume and masks real (material) changes.

What is material and what is not will vary on a site and page basis. That is, it cannot be simplistically specified.

The volume of new and updated content

As the following graphs show, the number of unique responses and the volume of those responses generated by web sites decline over time.

These graphs were generated from data created by running the LogSummary program (HREF14) against web server logs. A discussion of the information from the NASA and Clarknet sites and summarised statistics used to generate the graphs are provided by Arlitt and Williamson, 1996 (HREF19).

The LogSummary program reads web server logs and attempts to identify unique responses based on the requested URL and the length of the response. Web server logs do not contain enough information to produce a completely accurate estimate, but nevertheless, the trends are interesting. Specific issues with log processing include:

- As the actual bytes of the response are not available, the program can't really know whether a response is novel, so it guesses based on the URL and content-length (as logged).
- POSTed (request) data is not available in the logs. Hence, different responses for the "same" URL may often be caused by it processing different POSTed data. In this
case, an "update" is counted when it is more properly classified as a "new" request/response (that is, the POSTed data should be appended to the url, as is GETed data). This doesn't change the "same/not-same" split, but it can disconcertingly inflate the "updated" data statistics.

- Identical content with different response lengths caused by differences in HTTP/1.0 and HTTP/1.1 responses (and in particular, response chunking which alters the response length as logged) is counted as "updated".
- Aborted responses (caused by the user clicking the STOP button in the browser, or navigating to another page before the response is completely received) are not indicated in the log and hence appear to be updated, shorter responses, considerably inflating the number of updates.
- Genuinely different content with the same URL and response length is not identified as "updated".

Hence, the output of LogSummary must be taken with a large grain of salt. It will typically overestimate the amount of updated content, sometimes quite dramatically.

Here's a summary of the logs analysed to produce the following graphs:

<table>
<thead>
<tr>
<th>Site</th>
<th>Date Range</th>
<th>Total Successful (Resp 200) Requests</th>
<th>Distinct request URLs</th>
<th>&quot;Updated&quot; responses</th>
<th>Average response size (bytes)</th>
<th>Average first response size (bytes)</th>
<th>Average updated response size (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NASA</td>
<td>1 Jul 1995 - 31 Aug 1995</td>
<td>3,100,360</td>
<td>9,362</td>
<td>21,738</td>
<td>21,224</td>
<td>27,024</td>
<td>90,440</td>
</tr>
<tr>
<td>Clarknet</td>
<td>28 Aug 1995 - 10 Sep 1995</td>
<td>2,950,017</td>
<td>32,909</td>
<td>17,150</td>
<td>9,838</td>
<td>13,218</td>
<td>8,921</td>
</tr>
<tr>
<td>Large Public Australian Site</td>
<td>5 Nov 2002 - 1 Dec 2002</td>
<td>4,860,626</td>
<td>259,819</td>
<td>39,663</td>
<td>10,277</td>
<td>37,056</td>
<td>66,686</td>
</tr>
</tbody>
</table>

Legend:

- Total Successful (Resp 200) Requests: the number of requests receiving a HTTP 200 (successful) response code
- "Distinct request URLs": the number of distinct URL's (including request line parameters) found in the logs
- "Updated" responses: the number of requests for a previously seen URL but with a different response length
- Average response size (bytes): the average length of all successful requests
- Average first response size (bytes): the average length of all responses seen for the first time in the logs. That is, excludes the lengths of updated and repeated responses for a URL.
- Average updated response size (bytes): the average length of updated responses. That is, excludes the lengths of the first response and identical (repeated) responses.

**NASA log**

The logs available from The Internet Traffic Archive ([HREF18](http://ausweb.scu.edu.au/aw03/papers/fitch/paper.html)) covered 1 Jul 1995 - 31 Aug 1995 and contained a total of 3,461,612 requests. Of these 61,012 were cgi-bin requests of which 60,465 were imagemap processing requests almost all of which resulted in HTTP response code 302 (redirection).
Clarknet ISP

Clarknet is a large ISP in the Metro Baltimore-Washington DC area. The logs, also available from The Internet Traffic Archive (HREF18), covered 28 Aug 1995 - 10 Sep 1995 and contained a total 3,328,632 requests. Of these 51,747 were cgi-bin requests of which 42,543 resulted in HTTP response code 200 (successful).

Large Public Australian Site

This public Australian site has a very large document and image base. The logs processed covered 5 Nov 2002 - 1 Dec 2002, and contained a total of 7,077,941 requests. Just over 1% of responses were dynamically generated responses to searches and other queries.
The final graph shows considerably higher numbers for unique response numbers and volumes:

- Whereas unique response numbers for NASA and Clarknet trend to below 1%, this site trends to between 1 and 2%
- Whereas unique response volumes for NASA and Clarknet trend to below 2%, this site trends to between 10 and 20%

Some of the reasons for these differences include:

1. This site has a high degree of dynamic content, and a much larger static document/image base which results in a slower decline in update volumes.
2. This site contains many large documents, increasing the likelihood of aborted responses generating a "false" unique (updated) response.
3. These logs (being from the year 2002) include both HTTP/1.0 and HTTP/1.1 clients. The HTTP/1.1 protocol supports response chunking which affects response size as logged, causing responses logically identical but which have different chunking to have different response lengths and hence be counted as unique responses.

**Approaches to recording all materially different responses**

We considered three approaches to recording all materially different responses generated by a web site:

1. using a network level sniffer to capture responses and process them asynchronously to the operation of the web server, on separate hardware
2. using a HTTP level proxy between clients and the server - in effect, providing a "reverse proxy"
3. inserting a filter into the web-server's input (request) and output (response) flow

The first two options have the advantage of being loosely coupled with the web server and its software, but suffer the fatal flaw of not being able to capture HTTPS (SSL) responses. The second option could conceivably capture HTTPS responses, but at the very high cost of compromising the end-to-end nature expected of SSL sessions. This option also suffers from the additional problems of increasing the latency of the response (due to the extra TCP/IP connection) and the masking of the end IP address (which is used by many server-side mechanisms, from logging to authentication).

The third option provides full access to the request and response and is well supported by the modern architectures of Apache 2 and Microsoft's IIS. However, its location in the critical path of the request and response requires careful design.
The system we've implemented, pageVault (HREF6), takes this third approach. Its design attempts to address the problems of recording all materially different responses as described below.

**The architecture of pageVault**

PageVault is composed of 4 components:

a. **Filter**

Runs inside the web server's address space, identifying potentially unique request/response pairs and writing them to disk. Because the filter does not have a global view of the web server (many web servers are at least multi-processing and often run across several machines at different locations), and only maintains a local and recent history of what responses have been sent, it will generate some "false positives": request/response pairs that are not really unique. These false positives are identified and removed in subsequent components.

b. **Distributor**

Runs as a separate process, usually on the same machine as the web server, reading the temporary disk files of potentially unique responses generated by the filter. The distributor is able to immediately identify most of the false positives generated by the filter, removing them from further processing. The request URL and checksum of the remainder are sent to the archiver component, and if deemed unique by the archiver, the distributor compresses the response and sends it to the archiver.

If required, the distributor can route responses to separate archivers based on characteristics of the request.

c. **Archiver**

Runs as a separate process, usually on a separate machine from the web server/filter/distributor. The archiver maintains a persistent database of archived requests. The database architecture is a simple but extremely efficient and scalable B+Tree based on the open source JDBM software (HREF13).

When sent a request URL and checksum from the distributor, the archiver uses this database to determine if the request/response is unique. If it is unique, it solicits the complete details from the distributor and stores it in the archiver database.

The archiver also exposes a query interface used by the query servlet component to search and retrieve from the archive.

Because the archiver can process responses from multiple distributors, this architecture lends itself to the establishment of web response (or electronic document) notaries and to "federated" or "union" archives of web content.

All such an archive requires is that the web sites of interest run the Filter and Distributor components, and that the Distributor is configured to use the notary's or federated archive's Archiver component.

d. **Query servlet**

Runs as a servlet in a Java Servlet framework, such as Tomcat or Jetty. Provides a search and retrieval frontend to the archiver's database.
The filter is the most critical component of the system, running within the web server's address space and hooking into the request-response path.

Whilst Apache 2 and Microsoft's IIS web server have architectures that support filtering, the world's most widely used server, Apache version 1.x, does not support generalised filtering without radical changes or potentially significant performance impacts. Hence, a pageVault filter for Apache version 1.x is currently not planned.

Some of the key decisions made in the design of the pageVault architecture were:

- Place as little code in the filter as possible. This facilitates robustness and porting across server architectures and versions.

- The pageVault filter should be able to reject responses which are of no interest as early as possible, based on request URL (starting or ending strings or regular expression matches) or content type.

- It must be easy for non-material content to be defined and the filter must efficiently ignore such content when deciding whether the request/response is unique. For example, personalisation of web pages will often result in non-material differences between responses. PageVault allows any number of "signatures" of non-material content to be defined on a response (or set of responses based on URL matching of starting or ending strings or regular expressions) using starting and ending tokens which mark the boundaries of the non-material content.

For example, a site could define that all pages with a URL matching the regular expression /public/products/.*\.asp contain non-material content between these tokens:

- "Shopping Cart summary:" and "</TD>"
Then, when a response with a URL of, say /public/products/widget128.asp is being processed containing, say:

```html
<HTML>
....
<BODY>
<TABLE>
 <TD><H2>Widgets R Us Product Catalogue</H2></TD>
 <TD><B>Shopping Cart summary: Your cart has 4 items totalling $12.50</B>
 <A HREF="/showCart?id=1234">(details...)</A></TD>
</TR>
 <TR>
 <TD><B>Widget 128 - just what you need</B>
 ...</TD>
 <TD>Logged on as User: fred at 12 April 2003.</TD>
</TR>
...</BODY>
</HTML>
```

The two strings: "Your cart has 4 items totalling $12.50" <A HREF="/showCart?id=1234">(details...)</A> and "fred at 12 April 2003" would be excluded from the contents which are checksummed and hence used to determine response uniqueness. Hence, pageVault would calculate the same checksum for this URL regardless of the current contents of the shopping cart, the current user and the current date.

- The pageVault filter should be configurable at the "virtual server" level rather than at a global server level, as it is quite likely that web site administrators will want to determine archival settings at this level.

- Optimisation of the filter code is vital to good performance. The additional processing load imposed by the pageVault filter must be inconsequential for almost all sites.

- A single pageVault archive should be able to contain content collected from several web servers of arbitrary architectures. Many organisations run multiple web servers, and an organisation wide view requires that the content from every server be viewed coherently. The archiver may be run by a separate organisation as either an aggregation or notary service.

Additionally, services which currently attempt to harvest content from affiliated web sites may want to use pageVault to construct a unified database of content. This would provide a more complete coverage than spidering at a fraction of the bandwidth cost and a more easily accessible archive.

- Responses from a single web server should be able to be sent to different archives based on request URL (starting or ending strings or regular expression matches).

- The pageVault archive must support the complete viewing experience of the archived site(s) at a "point in time". That is, users of the archive must be able to navigate through the site as if it was "live". Javascript, stylesheets, applets and other client side objects must be delivered to the browser exactly as they were originally at that time.

Users of the archive must be able to search for starting pages using a combination of URL (including pattern matching), date and time ranges and frequency of update. Anticipated questions of the archive included:
what did this page look like at 9:30AM on the 5th May last year?

exactly how many times and how has this particular page changed over the 6 months?

exactly how did we respond to search requests containing the word "poison" over the last 18 months?

which images in the "logos" subdirectory have changed this week, and how?

the content of this URL has been corrupted - I need to recover it to the version as of August 17th - what did it look like then?

Side-by-side comparisons of different responses to the same request must be available, along with metadata recording the date/time of responses, the perceived client IP address and HTTP response header data.

- PageVault must not require any changes to any existing web based applications or web server software. It should be simple to understand, install and manage.

PageVault does not address all the issues in this problem domain. Specifically:

1. PageVault cannot answer questions such as why content changed or who changed it. Use of change management/versioning systems must be enforced to record such information.

2. The current version of pageVault does not support the free text searching of the archive. Hence queries such as "show me all unique responses in this part of the web site containing the text health and safety published in the last 2 months" cannot be answered. This facility is a planned enhancement.

3. PageVault does not archive content that is never viewed. The universe of unviewed content, is of course infinite (eg, search responses for every possible search string), but the fact that pageVault will not archive static pages that are never visited may be slightly disconcerting. In practice, external web spiders tend to visit all crawlable pages, and it is trivial to run spiders to crawl intranet pages to "prime the pump", so the problem may be moot.

4. WAP etc users may receive radically different content sent in response to the same URL based on server processing of client capabilities. In these cases, the responses recorded by pageVault will appear to "flap" between two or more different versions. Although each version will only be stored once in the archive, the pageVault archive index will add a pointer to these versions each time that the "flapping" response is generated. A future version of pageVault may address this issue by including specific client characteristics as part of the archived request URL.

5. PageVault cannot determine who saw what content unless the identity of the viewer is stored in or derivable from either the content of the request or the response. It can only determine what content was delivered and when. A future enhancement will support the correlation of web server logs with the pageVault archive, but issues caused by mapping of users to IP addresses and the actions of proxies and caches make exact matching of viewers to delivered content extremely problematic (HREF7).

**Performance impact of pageVault on the Apache 2 server**

Effective additional load on the webserver caused by the checksum calculation on responses is difficult to gauge because the IO and CPU resources required to generate a page vary enormously depending on its method of production (eg, straight copy from disk compared to labyrinthine database calls, server side includes, stylesheet processing etc), whereas the pageVault overhead is a small constant amount per request and per byte of response data checksummed. However, benchmarks performed on an "out of the box" configuration of Apache 2.0.40 running on a 750MHz Sparc architecture under Solaris 8
(prefork MPM, 1 client) indicate the pageVault filter adds a CPU overhead of approximately 0.2 milliseconds per request plus 0.2 milliseconds per 10KB of response generated. By way of comparison, the simplest possible delivery of a static image file from disk using the same configuration without pageVault requires approximately 1.1 milliseconds plus 0.02 milliseconds per 10KB of response generated.

With an "average" simple static web response of 10KB, the total service time with pageVault increases from around 1.1 milliseconds to around 1.5 milliseconds. However, in practice many commonly received responses are dynamically generated. Simple static responses tend to be cached more often and hence are requested less frequently, and as observed above, there is an increasing trend for pages to be generated via script. Rather than taking around 1 milliseconds per page, response generated by script typically take several milliseconds and often tens of milliseconds. Under these conditions, the pageVault overhead will almost always be insignificant.

The string matching algorithms for finding content to exclude from the unique response calculation are highly optimized and testing reveals negligible overhead. That is, the cost of searching for exclusion strings is insignificant compared to the cost of the checksum calculation.

PageVault uses server memory for a per-task hash table of previously seen responses and checksums and for a response buffer area.

The per-task response/checksum hash table is optimized for space and CPU time and with the default hash table size of 511 entries, the per task memory requirement is 12K. When run in threaded mode, each thread in a task shares this hash table. The hash table implementation supports multiple concurrent readers and writers without requiring locking, at the cost of the occasional "false positives" which are later identified and removed.

A pageVault response buffer is used to cache as much of a response as possible whilst it is being generated and transmitted, should that response prove to be unique and hence require archiving. Only when the response is complete can the checksum be evaluated for uniqueness. Large responses will overflow the pageVault buffer (set by default to 48K), and hence force the response to be written to disk, from where it must be deleted if (as is commonly the case) it proves to be not unique. So there is a tradeoff between CPU and memory in the setting of the pageVault response buffer parameter.

On an uncommonly busy 100 task, non-threading server the default memory overhead of pageVault within the web server's address space is 100 * 12K = 1.2MB plus 50K for each concurrent request (48K buffer plus 2K for other data structures), giving a total of 6.2MB for 100 concurrent requests. File I/O buffers allocated when disk files containing possibly unique responses need to be written will temporarily increase the memory load.

The current status of pageVault (May 2003)

PageVault filters for the Apache 2 web server and Microsoft IIS Versions 4 and 5 have been implemented. Although filtering is technically possible with Apache 1.x, because of the issues with the non-standard techniques used to achieve this and the broad Apache 1.x code base, no general version for Apache 1.x is planned.

Load testing with an archive size of 1 million responses has validated the basic archive architecture with no noticeable increase in record insertion or retrieval time, with the JDBM B+Tree indices performing very well.

Full text searching of the archive contents will be implemented by August 2003, followed by correlation between the archive and server access logs which will enable weak linkages between users and accessed versions of content ('weak' due to the well-documented issues associated with correlating HTTP requests and identifying users (HREF7)).
Conclusions

With the critical role played by web servers in corporate communication and the recognition that information published on a web site has the same legal status as its paper equivalent, knowing exactly what has been delivered to viewers of a web site is as much a necessity as keeping file copies of official paper correspondence.

Current methods of establishing within context what was published at a point in time have such severe problems and limitations that they cannot be relied upon for a general solution. However, tracking and archiving changed content as it is generated and delivered is an efficient and effective approach that has been validated by the pageVault implementation.

As well as providing an attractive archiving solution for individual websites, pageVault also supports the creation of "union" archives and hence offers a cost-effective alternative to multi-site harvesting by spiders.

References

HREF1  
Netcraft Web Server Survey for December 2002  
[http://www.netcraft.com/Survey/index-200212.html]

HREF2  
Towards An Australian Strategy for the Information Economy - COMMONWEALTH GOVERNMENT LEADS THE WAY", Media Release from Minister for Finance and Administration  

HREF3  
Implementation of the Government Paperwork Elimination Act (GPEA)  
[http://www.whitehouse.gov/omb/fedreg/gpea2.html]

HREF4  
Web Content Meets Records Management - Diane Marsili, e-doc magazine  

HREF5  
Legal Aspects of online access and digital archives - Martin von Haller Gronbaek  

HREF6  
pageVault home page  

HREF7  
Analog: How the web works - Stephen Turner  
[http://www.analog.cx/docs/webworks.html]

HREF8  
A policy for keeping records of web-based activity in the Commonwealth Government, National Archives of Australia  

HREF9  
Archiving the web: The national collection of Australian online publications - Margaret Phillips, National Library of Australia  

HREF10  
Minerva - The Library of Congress  
[http://www.loc.gov/minerva/]

HREF11  
Access to Web Archives: the Nordic Web Archive access project - Svein Arne Brygfjeld National Library of Norway  
[http://www.analog.cx/docs/webworks.html]
Web site archiving – an approach to recording every materially different response produced by a website

[HREF12 Brewster Kahle's Internet Archive/The Wayback Machine [http://www.archive.org/]

[HREF13 JDBM - open source B+Tree implementation [http://jdbm.sourceforge.net/]


[HREF16 Assent is the Key to Valid Click-Through Agreements - W. Scott Petty, King & Spalding [http://www.kslaw.com/library/articles.asp?959]

[HREF17 Enhancing the Enforceability of Online Terms - Goodwin Procter [http://www.goodwinprocter.com/publications/IPA_enforceability_3_02.pdf]

[HREF18 Internet Traffic Archive [http://ita.ee.lbl.gov]


About the author

Kent Fitch has worked as a programmer for over 20 years. Trained in Unix at UNSW in the 1970's, he has worked in applications, database, networks, systems programming using a wide variety of tools. Since 1983 he has been a principal of the 3 person Canberra software development company, Project Computing Pty Ltd. He has developed many commercial systems and communications packages and custom software for many clients. Since 1993, he has been developing software for web sites and currently specialises in Java and C programming, applications of XML and RDF/Topic Maps and web based user interfaces.

Copyright

Kent Fitch, © 2003. The author assigns to Southern Cross University and other educational and non-profit institutions a non-exclusive licence to use this document for personal use and in courses of instruction provided that the article is used in full and this copyright statement is reproduced. The author also grants a non-exclusive licence to Southern Cross University to publish this document in full on the World Wide Web and on CD-ROM and in printed form with the conference papers and for the document to be published on mirrors on the World Wide Web.