A FRAMEWORK FOR A WEBSITE SNAPSHOT MANAGEMENT SYSTEM

David Chao, San Francisco State University, dchao@sfsu.edu
Sam Gill, San Francisco State University, sgill@sfsu.edu

ABSTRACT

This paper presents a framework for a website snapshot management system. The objective of the system is to recreate webpage snapshots of every published webpage including their code and rendition upon users' requests. We define four levels of snapshots and design software components to create snapshots that meet the requirement of each level. A website may choose to maintain the appropriate level of snapshot that fits its needs.

Keywords: Website snapshot, management, framework

1. INTRODUCTION

A business enterprise operates in the context of a dynamic changing environment. Changes may originate externally such as changes in government regulations, enterprise partners, and enterprise technologies. Changes may also originate internally such as changes in enterprise mission, strategy, policies, structure, processes, and products and services portfolio. Consequently, an enterprise website may change constantly to reflect the dynamic nature of its environment resulting in changes in its website infrastructure, structure, and content.

Archival data can be generated as a result of changes to a web site. Examples of historical data are earlier versions of a current page and deleted pages. Historical data are valuable in supporting applications that require historical data, such as applications that perform analyses to study certain trends in the study subject, or answering questions about website content in the past for audit and compliance purposes. Websites may also be required to preserve historical data due to government or organizational policies. Allowing users to access historical data is a value-added service. Social websites such as MySpace let users view past personal blogs to enrich users' web experience. Preserving historical data is necessary to support these applications.

Recognizing the value of historical data, we consider a website as a system consisting of a current website where up-to-date web pages are published and, in parallel, a historical website where every version of current pages and deleted pages are kept. A version of a webpage is actually the page's snapshot from the time the page is first published to the time the page is modified or deleted. Therefore, maintaining website snapshots offers a solution to managing historical data. A website snapshot is the state of a web site at a point in time. By maintaining website snapshots, a user's request for historical data at a specific point in time can be retrieved from the snapshot at the time.

This paper proposes a framework for a website snapshot management system. The objective of the system is to recreate webpage snapshots of every published webpage including their code and rendition upon users' requests. Past research in the management of historical data has developed web page versioning schemes in retrieving or recreating previous versions of a web page [6, 7, 9, 10]. These schemes typically are designed to keep track the internal changes to a static web page and ignore the fact that many of today's web pages are created dynamically from many sources such as style sheets and content databases. These researches have also ignored the issue of preserving changes to website's infrastructure. In general, changes to the operating system, the web service, the database management system, and the server-side computer language used to create the server side pages, collectively known as the "stack", have not been considered in taking web page snapshots and will cause difficulty in re-rendering them when the infrastructure, consisting of the stack components, has changed. Another popular practice in preserving historical data is periodically creating date-time stamped archives of the website. This type of archives is unable to satisfy a user's request for historical data between archives.

The proposed framework considers the rendition of a web page to be the result of running code in many source files and the page may contain contents retrieved from databases. These source files and the databases may be internal and managed by the website, or external and managed by other websites. The framework saves changes to the web pages and website's infrastructure so that snapshots created by discontinued technologies can be rendered properly. Figure 1 presents an overview of the major components of the framework and the two external

clients it interfaces. The framework consists of a Website Changes Tracking System for preserving website changes, an Inter-Website Snapshot Exchange System for exchanging snapshots between websites, a Snapshot Request Handling System for processing user's requests for historical data and a Web Server Virtualization System for supporting older stacks.

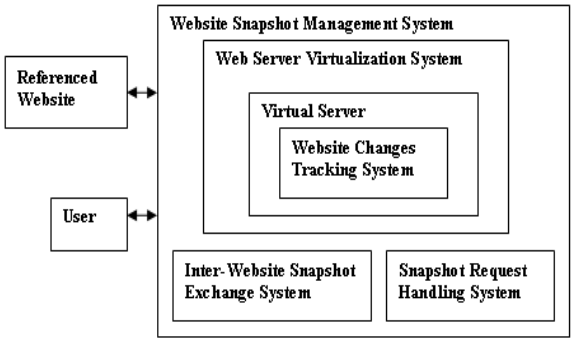

Figure 1: An overview of the Website Snapshot Management System and its clients.

This paper is organized as follows. Section 2 presents an analysis of the snapshot management problem. Section 3 presents the design of the proposed framework. Section 4 concludes the paper.

2. PROBLEM ANALYSIS

2.1 Impact of Website Changes in Generating Historical Data

A web site is a system of integrated web pages embedded with links to reference related pages. At any given time, a Uniform Resource Locator (URL) [13] uniquely identifies a web page on the Internet. URLs are also known as the addresses, or paths, of web pages on the Internet. The relationship between URL and web page is m:m where each URL may be associated with many web pages and each web page may be associated with many URLs over time. A URL may point to a web page that is different from the one it pointed to earlier. This may happen because of the renaming of web pages and directories.

The URLs of all web pages currently published by a web site are web site's *current links*. A web page is considered current since the time it is published until the time it is changed when the old version becomes the page's snapshot between the published time and the update time, and the update time becomes the new version's published time. In a sense, the current version of a web page is the page's snapshot since its published time. A web page may be changed repetitively. If snapshots are kept in an archive, the combined value of a page's URL and published time can be used as a *snapshot - ID* to uniquely identify a snapshot.

Table 1 summarizes the impact of a web site reorganization and change to a web page in generating historical resources; it assumes the existence of an archive where snapshots of web pages are stored. When a web site reorganizes its directory structure, all impacted web pages will have a new URL. Similarly, when a web page is renamed or relocated to other directory, it will have a new URL as well. When a web page is modified, its URL will not change and the old version becomes a snapshot and is archived. A deleted document is archived as well for later retrieval.

The URLs invalidated by a web site due to reorganization, web page removal, renaming, or relocation, plus all *snapshot -ID* s are a web site's *historical links*. Maintaining historical links enables a web site to retrieve the web page associated with an out-dated URL submitted by users, deleted web pages, and web page snapshots. This will improve a web site's ability in searching documents and provide needed historical information for users.

Table 1: Impact of website reorganization and web page changes

Web site actions	Impact on current and historical links
Adding a new web page	Adding a new URL to current links
Modifying a web page	No change to URL; the old web page becomes a snapshot and archived
Deleting a web page	Deleting a current link; adding a historical link and web page is archived
Renaming a web page	Adding a new URL to current links; the old URL becomes historical link
Relocating a web page	Adding a new URL to current links; the old URL becomes historical link
Reorganization	Adding all affected web pages' URLs to current links; adding all affected web pages' old URLs become historical

2.2 Factors Affecting the State of a Web Page

As discussed earlier, a web page typically consists of static and dynamic contents that are constructed from internal resources and external resources. The state of a web page captures not only the code that defines the web page but also the presentation of the web page as a results of running the code. Factors affecting the rendering of web page include: (1) Web page code: This is the code defined within the web page that creates web page's static and dynamic contents including server-side and/or client-side scripts. (2) The state of internal resources it references. (3) The state of external resources it references. (4) Server-side and client-side technological factors: A web page is often created by using many technologies and may reference documents that are created by a different set of technologies. The complete rendering of a web page requires that all technologies, client-side and server-side, are properly implemented.

The first three factors define the content of a web page; they are *content* factors. The last factor is *technological* factor. Based on the content factors' effects, three levels of a web page's snapshot can be defined. (1) Level 1 snapshot: A web page snapshot is the state of the web page code at snapshot time. (2) Level 2 snapshot: A Level 2 snapshot is a Level 1 snapshot with the additional requirement that all the internal resources it references are Level 1 snapshot at the snapshot time. These internal resources can be categorized into two groups: database and non-database files. For a database file to be a Level 1 snapshot, it must be consistent with the state of the database at the snapshot time. (3) Level 3 snapshot: A Level 3 snapshot is a Level 2 snapshot with the additional requirement that all the external resources it references are Level 2 snapshots at the snapshot time.

The state of the website technology at the snapshot time must be preserved in order to create the web page snapshots at the snapshot time. A web page snapshot will render completely if all relevant technologies are available; otherwise it will only render partially. We define two levels of technological factor: (1) Plus 0: If only some technologies at snapshot time are available. (2) Plus 1: If all technologies at snapshot time are available.

Table 2 summarizes all possible levels of snapshot states. The Level 1 and Level 2 snapshots involve internal resources only; the Level 3 snapshots involve external resources. Websites that are able to deliver Level 2 snapshots are snapshot-enabled websites. To create Level 3 snapshots all the referenced websites must be snapshot-enabled. The framework we present is designed to create Level 3 + 1 snapshots.

		Content Factor		
Technological Factor		Level 1	Level 2	Level 3
	Plus 0	Level 1 + 0	Level 2 + 0	Level 3 + 0
	Plus 1	Level 1 + 1	Level 2 + 1	Level 3 + 1

Table 2: Levels of website snapshot

3. DESIGN OF MAJOR COMPONENTS OF A WEBSITE SNAPSHOT MANAGEMENT SYSTEM

This section provides a description of the major components of a snapshot-enabled website snapshot.

3.1 Design of Website Changes Tracking System

This system consists of two modules: a Website Document Changes Tracking Manager and a Content Database Snapshot manager. The objectives of this system are to track and preserve changes, and deliver snapshots of web pages and content databases.

3.1.1 Design of Website Document Changes Tracking Manager

In the following discussions, the term *web document* represents either a web page or an internal resource referenced by a web page. Web documents are *files* to the website. Changes to a website include the additions, modifications, deletions, and relocations to web documents. A web document is identified by its URL which consists of host name, the path to the directory where it resides on the host, and the document's name. The host name typically represents the website's root directory, and the path is the folders to the web page.

We design a logging and archiving scheme for tracking website changes. The objectives of this system are: (1) Recording the time and types of changes made to web pages. (2) Recording the time and types of changes made to website structure. The system records the website structure changes by recording their impacts to web pages. For example, when a web folder is renamed all the web pages' URL in the folder will be renamed and their changes are recorded in the log. (3) Saving a copy all snapshots generated as results of changes and deletions to web documents. The scheme is originally published in [4] and we present an overview of the scheme.

Design of the logging and archiving scheme

The log, named TemporalURLLog, is designed to keep the history of changes to web documents. It has five fields: URL, PublishDate, DocExpireDate, URLExpireDate, and NewURL. The URL field records a document's URL; the PublishDate records the time the document is published; the DocExpireDate field records the time the document is modified or deleted; the URLExpireDate records the time document's URL expires; the NewURL field records the document's new URL should a change to the document creates a new URL for the document. The value of URL + PublishDate uniquely identifies a document's sanpshot. The Archive is a repository of deleted documents and document snapshots. Those archived documents are saved in the Archive using URL + PublishDate as file name.

With this scheme any web document's snapshot can be retrieved from the archive by using its original URL so that changes to the website's structure are logically preserved.

3.1.2 Design of Content Database Snapshot Manager

Content databases provide the dynamic contents for web pages. The objective of this system is to generate the snapshot of these databases. Database snapshot management has been well-studied since early 1980 [1] and has been implemented in many of today's database management systems such as Oracle and SQL Server [11, 12]. Users may define and use snapshots which are read-only and materialized copy of selected database that eventually become stale. At that time users may issue a refresh command to bring the snapshot to a new consistent state with the database. Changes to the database are recorded in a log. A typical design of the log is: Record's Key Field+ Record's Other Fields + Update Flag + Time Stamp. The Update Flag field indicates type of updates and the Time Stamp field records the time the record is updated. Treating a record's modification as the deletion of the before-image and insertion of the after-image, the update flag may have two values, I for insertion and D for deletion. Using the update log, net changes since the snapshot was created or last refreshed can be generated to support a differential refresh of the snapshots [2, 3, 12].

A dynamic web page typically consists of many placeholders for contents of a specific subject. Each placeholder may hold one or more data items retrieved from possibly many data sources. There are many types of databases that provide contents to web pages. Some of them are created to support day-to-day business operations or decision makings. They are designed to model business entities such as customers, orders, products and are independent of the web pages that reference them. Some databases are created specifically as storage to hold contents of placeholders on a web page. For example, a small table may be used to store the names of fruits used for creating a dropdown list on a web page. They are page-oriented content databases. Tracking changes to the page-oriented content databases depends on the design of these page-oriented databases. They may be classified into three types:

(1) Site-level content database: A site-level content database uses one table to hold data for placeholders of many or all web pages of a website. Assuming the website has many web pages each with a unique identifier, PageID, and each web page may have many placeholders each with a unique identifier, PlaceHolderID, such as placeholder's name, then a generic design for a site-level content database will have the following fields: PageID, PlaceHolderID, and Content, where the Content field stores one item of data for the placeholder. Accordingly, the log recording the changes of the database, named ContentHistoryLog, will be: PageID, PlaceHolderID, Content, UpdateFlag, TimeStamp.

(2) Page-Level Content Database: A page-level content database uses one table to hold data for many or all placeholders of a webpage. The design will be: PlaceHolderID, Content. The ContentHistoryLog of it will be: PlaceHolderID, Content, UpdateFlag, TimeStamp.

(3) Placeholder-Level Content Database: A placeholder-level content database uses one table to hold data for only one placeholder of a webpage. The content database will simply be: Content and Content will be the key. The ContentHistoryLog of it will be: Content, UpdateFlag, TimeStamp.

Generating content database snapshot Using the ContentHistoryLog, changes to the content database between any points in time can be generated. Assuming the content database is cached at time T1 and the content database snapshot at time T2 is requested where T2 > T1, then the snapshot can be recreated by two ways: (1) Apply the changes between T2 and the current time to the current content database in a backward-refresh. (2) Apply the changes between T1 and T2 to the cached content database in a forward-refresh.

3.2 Design of Inter-Website Snapshot Exchange System

The objective of this system is to facilitate the exchange of snapshots between websites to obtain the snapshots of external resources. Internet standards such as web service [14] can be deployed to achieve this objective. A web service is a way of serving a request from a client and sending the results generated by a server to the client over the internet. We propose two web services, one for exchange web document snapshot and the other for exchange content database snapshot.

RequestWebDocumentSnapshot service: Unlike requesting a current web resource, requesting a snapshot of an external resource both the URL of the referenced document and the snapshot time must submit to the website. Note that every web document snapshot in the Archive uses its URL + PublishDate as file name, so the PublishDate is the snapshot time of the referenced document. Given the URL and the snapshot time, the snapshot of a web page can be retrieved by processing the TemporalURLLog and the Archive. This web service takes a valid URL and the snapshot time as inputs and returns the link to snapshot at the specified time.

RequestContentDBSnapshot service: A typical way of retrieving dynamic content from a database is by embedding a SQL command in the script code of the dynamic web page. For example, to search for products below $100, a SQL command "SELECT * FROM *product* WHERE price < 100;" may be embedded. This command must be changed to read the snapshot of the *product* table. Essentially, the Content Database Snapshot System must know the SQL query the snapshot time. This can be done by adding snapshot time to the SQL command. For example, to access the snapshot of the *product* table on 12/25/07, the SQL statement is modified to: "SELECT * FROM *product* WHERE price < 100 AS OF *12/25/07*;" This web service is designed for external websites to request a snapshot based on a SQL command and the snapshot time. It returns the name of the snapshot created by the Content Database Snapshot System.

3.3 Design of Web Server Virtualization System

The objective of this system is to preserve the infrastructure (the stack) used to create and run the web pages archived in the historical website. Two examples of the components of a stack that constitute an infrastructure are the Microsoft stack with Windows, Internet Information Service (IIS), SQL Server, and a .NET language; and the LAMP stack with Linux, Apache, MySQL, and PHP. A stack becomes dated when any one of its components is upgraded and as result rendering an old page on a new stack cannot guarantee that the old page would be rendered identically as if it were being rendered on the old stack. Originally, server virtualization is defined as the capability of using one physical server to emulate many servers with virtual machines where each may run a different operating system [5]. Server virtualization has gained momentum in the effort to consolidate multiple physical servers onto one physical server running virtualized versions of the disparate physical servers. This framework extends the server virtualization technology to the preservation of discontinued or upgraded infrastructure.

We consider a web server, W, as a system defined by the four components of the stack: the host operating system, O, the web service, S, the database management system, D, and the server side web language, L. Among these four factors, the operating system is a dominating factor that determines the rest of the components of the stack that can be implemented. The state of a web server is denoted as $W_i(O_j, S_k, D_l, L_m)$, where O_j is the operating system version on serve W_i, S_k is the web service version, D_l is the database management system version, and L_m is the language version. Implementing changes to any of these four components of the stack will cause web pages that were created before the code change obsolete. Changes in the stack trigger the creation of a new virtual server where the state of the web server, W_i, before adopting the changes is preserved as a virtual server. Conditions triggering the virtualization are discussed below assuming $W_i(O_j, S_k, D_l, L_m)$ is the current state of the web server:

Change in L: If change in language causes some web pages in $W_i(O_j, S_k, D_l, L_m)$ unable to run, then the web server enters a new state $W_{i+1}(O_j, S_k, D_l, L_{m+1})$ and the $W_i(O_j, S_k, D_l, L_m)$ is preserved.
Change in D: Similarly, If change in database management sytem causes some web pages in $W_i(O_j, S_k, D_l, L_m)$ unable to run, then the web server enters a new state $W_{i+1}(O_j, S_k, D_{l+1}, L_m)$ and the $W_i(O_j, S_k, D_l, L_m)$ is preserved.
Change in S: When the web service is changed or upgraded, the previous server is preserved and the web server enters a new state $W_{i+1}(O_j, S_{k+1}, D_l, L_m))$ and the $W_i(O_j, S_k, D_l, L_m)$ is preserved.
Change in O: When the operating system is upgraded, the previous server is preserved, and the web server enters a new state $W_{i+1}(O_{j+1}, S_k, D_l, L_m)$ and the old state $W_i(O_j, S_k, D_l, L_m)$ is preserved.

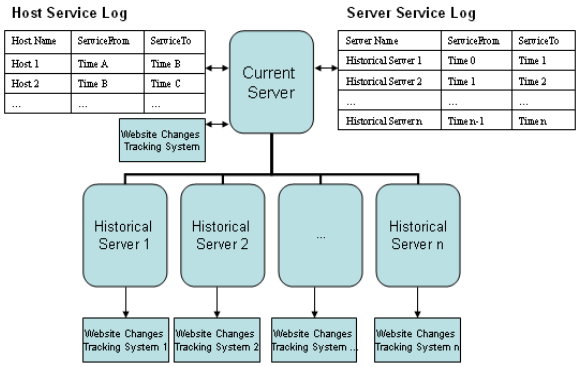

Figure 2: Design of virtualized web server.

Note that we assume the existence of a collection of virtual servers running on a host farm. Figure 2 shows the structure of the current host. It consists of many virtualized historical servers and a current server. Two data structures are used to record servers' service time. The Host Service Log records the time span each host serving as the current host at the time. The Server Service Log records the time span each virtual server serving as the current server. The current server and the historical servers are associated with a Website Changes Tracking System recording changes during the time it is the active current server. A request for a snapshot with a given snapshot time can be served first by determining the host and the server at the time and then retrieved from the historical server's Website Changes Tracking System.

3.4 Design of Snapshot Request Handling System

The objective of this system is to provide an interface for users to submit requests for snapshots and recreate the requested snapshots to users. It uses a keyword-based and URL-based searching for requests with specified snapshot time. The snapshot time will help to identify the historical server that was active at the time and searching is performed only at the historical server.

Keyword-based searching for snapshots Keyword-based server searching such as Google's Desktop [8] is freely available. To extend the technology to searching snapshots, a website must provide interface for users to enter both keywords and snapshot time. The searching targets the Archive of the Website Changes Tracking System of the historical server. All the webpage snapshots in the Archive have the page's published date as part of the filename. Archived files with published date greater than the snapshot time do not exist at the snapshot time. The search algorithm returns only relevant files with published date less than or equal to the snapshot time to users.

URL-based searching for snapshots The URL-based searching for snapshot can be implemented by submitting the URL with a specified snapshot time. There are two scenarios where this searching is practical: 1. the user would like to view the snapshot of a current page at a snapshot time. Hence the URL is current and valid. 2. The URL is not current and the user knows it was valid at a time in the past and would like to view the page at the time. This is a common experience of internet users to receive a file-not-found error after submitting an out-of-date URL from browser's Favorites or Bookmarks, or an URL from a book. We design two web services for these two scenarios:

RetrieveCurrentPageSnapshotAsOf service: This service takes a current URL and the snapshot time as inputs and returns the link to the snapshot at the specified time. The search is applied to the TemporalURLLog. Because the page may have a different URL earlier the search should perform in a *trace back* manner starting from the page's current entry in the TemporalURLLog and may continue to the historical web servers at the snapshot time. If the current URL's PublishDate is less than specified time, then the current document itself is its snapshot at the time.

RetrieveOldPageAsOf service: This service takes an out-of-date URL and the time it was valid as inputs and returns the link to the web page at the time. The search first determines the historical web server at the time, then the search is applied to the TemporalURLLog of the historical web serve rto locate the log entry with the same URL and the snapshot time is between the entry's PublishDate and its DocExpireDate. Once the snapshot entry is found, the snapshot itself is found in the Archive with a file name of URL + snapshot entry's PublishDate.

4. SUMMARY

This paper presents a framework for a website snapshot management system. The objective is to manage a website's historical data for effective use. We define four levels of snapshots and design software components to create snapshots that meet the requirement of each level. With the Website Document Changes Tracking Manager a website is able to generate Level 1 snapshot that is consistent with the web page code at the snapshot time. With the addition of the Content Database Snapshot

Manager a web site is able to generate Level 2 snapshot that is consistent with the web page's dynamic content retrieved from an internal database at the snapshot time. With the addition of the Inter-Website Snapshot Exchange system a web site is able to generate Level 3 snapshot that is consistent with the web page's external resources at the snapshot time. With the addition of the Web Server Virtualization System a web site is able to generate Level 3 + 1 snapshot that is consistent with the web page's rendition at the snapshot time. A website may choose to maintain the appropriate level of snapshot that fits its needs.

REFERENCES

1 Adiba, M. & Lindsay, B. (1980). Database snapshots. Proceedings of the 6th International Conference on Very Large Data Bases, pp. 86-91.

2 Blakeley, J. A, Larson, P. A. and Tompa, F. W., "Effiently Updating Materialized Views". In Proc.of the 1986 ACM SIGMOD International Conference, pages 61{71, August 1986.

3 Chao, D., Diehr, G., & Saharia, A., "A Differential Refresh Scheme for Remote End-User's Views," Journal of International Information Management, Spring 1996, Volume 5, Number 1, pp. 75-85.

4 Chao, David, Gill, Sam, "The Designing of Web Services to Deliver Web Documents Associated with Historical Links", Journal of Digital Information Management, Volume 4, Number 1, March 2006

5 Chappell, David (2007). Virtualization for Windows: A Technology Review. Microsoft White Paper

6 Chien, S., Tsotras, V., & Zaniolo, C. (2001) Efficient Management of Multiversion Documents by Object Referencing. Proceedings of 13th International Conference on Very large Data Bases, 2001.

7 Dyreson, C., Lin, H., Wang, Y. (2004). Managing Versions of Web Documents in a Transaction-time Web Server, The thirteenth world wide web conference, New York, 2004

8 Google Answers: Q Cache Engine Questions, http://answers.google.com/answers/main?cmd=threadview&id=237111

9 Haake, A., Hicks, D. (1996) VerSE: Towards Hypertext Versioning Styles. Proceedings of Hypertext 1996.

10 Marian, A., Gregory Cobena, S., & Mignet, L (2001) Change-Centric management of Versions in an XML Warehouse. Proceedings of the 27th VLDB Conference, Rome, Italy, 2001.

11 Oracle9i Materialized Views: An Oracle White Paper, May 2001

12 Tripp, K., "SQL Server 2005 Beta 2 Snapshot Isolation", SQLskills.com, February 2005

13 Uniform Resource Locators (URL), RFC 1738: http://www.w3.org/Addressing/rfc1738.txt/

14 W3C Web Services Activity Home Page. http://www.w3.org/2002/ws/