The Design of a Web Document Snapshots Delivery System

David Chao

College of Business, San Francisco State University, San Francisco, CA 94132
dchao@sfsu.edu

ABSTRACT

A web document snapshot is a point-in-time capture of its code and the resulting presentation of executing the code. It is used as a way of electronically preserving historical information published in web documents enabling an organization to audit a web document's contents at a point in the past and perform business analyses with historical information recorded in it. It is also an archived copy of a web document when it is changed. This research develops a system to deliver snapshots of a web document's static and dynamic contents when it is requested. The system consists of a Database Snapshot Manager for providing database snapshots and a Web Document Snapshot Manager for providing web document snapshots. Algorithms supporting the two managers are presented.

Keywords: web document snapshot, website snapshot management

1. INTRODUCTION

A web document snapshot is the state of a web document at a point in time (snaptime).

It enables an organization to audit a web document's contents at snaptime and perform business analyses with historical information recorded in it. It is also an archived copy of a web document when it is changed [6]. Many organizations such as government are mandated to electronically archive official web documents [7] [8], a web document snapshots management system will help to meet that requirement.

However, what is the state of a web document? A web document has dual definitions. Physically, it is a text file containing code, such as HTML or XML that defines its contents and presentation. Logically, it is a web page as displayed with a browser. From a user's perspective, the rendering of a web document matters more than its source code. Considering the fact that the code of a dynamic web document may render different contents at different times, therefore, the state of a web document is a point-in-time capture of its code and the resulting presentation of executing the code.

Factors affecting the state of a web document include:

a. Web document code: This includes the code that creates web page's static and dynamic contents. Frequently code creating dynamic contents is stored separately in a different file as in the case of Server-Side-Include and Code-Behind technologies.

b. The state of internal resources it references: Internal resources are files managed by a web site and are available in creating the web site's contents. Typical examples of internal resources referenced by a web document that affect its rendering are style sheets, files embedded by a Server-Site Include directive, image files, script files, and databases. These internal resources may create dynamic contents that change every time the document is opened, and they are subject to change that consequently changes the web document's rendering. Many of these files are *internal supporting files* that are created for supporting a web document and are not for publishing individually.

c. The state of external resources it references: External resources are files not managed by the web site but can be referenced in creating the web site's contents. A web document may reference external style sheets, components, web services, or databases. These external resources are also subject to change.

d. Web site host environment variables: Script code may reference a host's system variables in creating dynamic contents. A typical example is using the system clock to get the current date and time. A web document that displays the current date and time is always in a new state.

Four levels of web document snapshot Based on the four factors affecting web document's rendering, four levels of a web document's snapshot can be defined.

Level 1 snapshot: A web document snapshot is the state of web document code at snaptime. Creating level 1 snapshot enables a web site to trace the changes to the web document code over time.

Level 2 snapshot: A level 2 snapshot is a level 1 snapshot with the additional requirement that all the internal resources it references are at least level 1 snapshots at the same snaptime. 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 its state at the snapshot time. For a non-database file to be a level 1 snapshot, it must meet the definition above.

Level 3 snapshot: A level 3 snapshot is a level 2 snapshot with the additional requirement that all the external resources it references are at least level 2 snapshots at the snaptime.

Level 4 snapshot: A level 4 snapshot is a level 3 snapshot with the additional requirement that all the web site host's environment variables are reset to their values at snaptime.

The level 1 and level 2 snapshots are related to files that are managed by the web site, and changes to these files can be recorded. The level 3 and level 4 snapshots involve resources that are not managed by the web site and it's difficult, if not impossible, for the web site to keep track of changes to these resources. This research develops a web document snapshot management system to deliver web documents' level 1 and level 2 snapshots. Thus, the system consists of two modules: Database Snapshot Manager for maintaining database snapshots and Web Document Snapshot Manager for maintaining web document snapshots. The system is designed to deliver snapshots with any snaptime requirement, and snapshots are created only when requested. The Database Snapshot Manager is presented first.

2. DATABASE SNAPSHOT MANAGER

Database snapshot management and related materialized view management have been the topics of much research [2] [3] [4] [5]. Many schemes have been developed in maintaining database snapshots. This research adopts a generalized scheme that will meet the requirement of web document snapshots. It is presented below.

The objective of this module is to provide a database snapshot at any snaptime requested by users. This requires recording all updates in a log. The log uses time stamp to record update time, and use flags to indicate deletions and insertions where a modification is treated as the deletion of the old version followed by an insertion of the new version. With such an update log available, rolling back the current database using updates with time stamp later than the snaptime can generate a snapshot at any snaptime.

An alternative scheme is creating snapshots at different intervals. These snapshots can be *refreshed* to a new snaptime by applying updates between the old snaptime and the new snaptime. The refresh operation can go forward if the new snaptime is later than the old snaptime, otherwise it can go backward. With the availability of the two refresh operations, the optimal way is to select the snapshot whose snaptime is closest to the new snapshot, and only use updates between the snaptimes of the selected snapshot and the new snapshot. Let UpdateLog(t_{min}) represent the update log whose minimum time stamp value is t_{min}; and let UpdateLog(t_a, t_b) represent the segment of the update log with time stamp between t_a and t_b; assume there are n snapshots created each with snaptime t_i represented by $S(t_i)$ where i assumes values between 1 and n; assume the new snapshot's snaptime is t_s, the algorithm GeneratDatabaseSnapshot takes t_s as input and returns $S(t_s)$:

Algorithm GenerateDatabaseSnapshot:
Inputs: n snapshots, update log, and snaptime
Output:Database snapshot at snaptime

If t_s, < t_{min} Then
  Unable to generate snapshot using this log
Else
  Select $S(t_i)$ where t_i has the min $|t_s, - t_i|$
    where i = 1 to n
  If t_i, < t_s Then
    $S(t_s) = S(t_i)$.RefreshForward(UpdateLog(t_i, t_s))
  Else
    $S(t_s) = S(t_i)$.RefreshBackward(UpdateLog(t_i, t_s))
  End if
End if

3. WEB DOCUMENT SNAPSHOT MANAGER

The objective of the Web Document Snapshot Manager is to generate level 2 snapshots for all internal non-database files including the supporting files. On the Internet, a Uniform Resource Locator (URL) uniquely identifies a web document. The snapshot manager is designed to let users to use a URL with a specified snaptime to retrieve the document's snapshot. An analysis of the problem is given below.

3.1 Problem analysis

At a specific point of time, a Uniform Resource Locator (URL) uniquely identifies a document on the Internet. At the web site, it translates to a physical path to the web document. Although there exists a 1:1 relationship between a URL and a web document at a given time, in the life of the document the relationship is M:M. In a document's life, it may have associated with many URLs. The web site may reorganize by changing its directory structure, hence changing the path to a document; a document may be renamed, or moved to a different directory. These changes will give the document a new URL. A URL may have associated with many documents also. It may point to a document that is different from the one it pointed to earlier. For instance, a document A may be renamed to C and another document B may be renamed to A; hence the URL originally pointing to A is now pointing to a different document. It may also happen when a web site removes a URL but later reinstates the URL for another document.

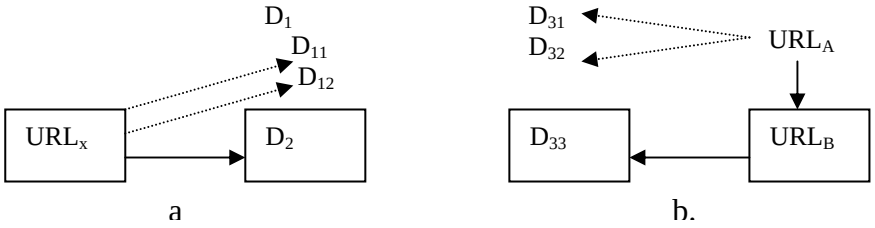

Figure 1: Relationship between URLs and web documents.

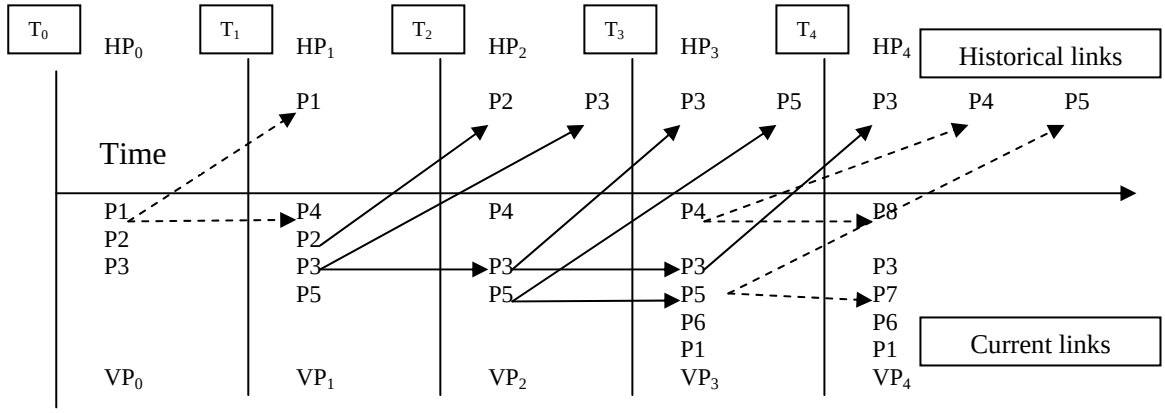

Figure 2: An example of the progression of a web site with changes since its initiation.

A web document may change its contents. Hence a URL may not only associate with many different documents, it may also associate with many versions of the same document. Figure 1 illustrates the relationship between URLs and web documents. Let D_{ij} represents document i, version j. In Figure 1a, URL_x initially points to D_{11} and D_{12}, then points to D_2. In Figure 1b, document D_3 initially associates with URL_A , then associates with URL_B.

URLs can become invalid and users may submit invalid URLs. In Figure 1b, URL_A changes to URL_B and points to D_{33}. Is the web site able to return D_3 if users submit URL_A ? Or is it able to return D_{32} it originally pointed to? Even valid URLs don't necessarily retrieve the documents users intend for it to. In Figure 1a, a user may use URL_x saved in browser's Favorits or Bookmarks trying to retrieve D_1. URLs invalidated by a web site due to reorganization, document removal, renaming, or relocation, plus the links to document snapshots are a web site's historical links. Tracking historical links by recording the time and the type of changes that occurred to a document and its URL enables a web site to retrieve the documents associated with the historical links.

Figure 2 illustrates the progression of a web site. T_0 is the time when the web site is initiated, and T_i denotes a point in time when its contents change. Period i is the interval of time between T_i and T_{i+1}. Assuming the path of a valid URL is unique within the web site, a valid path leads to a document currently published. In the sequel, paths and URLs are used interchangeably. There are two sets of paths in period i: VP_i , a set of all valid paths, and HP_i, a set of paths becoming historical at T_i. Note that HP_0 is null. In Figure 2, paths that produce two dotted arrow lines such as P1 in period 0, and P4 and P5 in period 3 represent documents that are renamed or relocated. In this case, only the document's path has changed but the document has not changed. Therefore, there is no need to archive any documents, but it is necessary to chain the old path to the new path in order to track the change. Paths that produce two solid arrow lines such as P3 in period 1, and P3 and P5 in period 2 represent documents that are modified. In this case, the path is still valid but the document before the change needs to be archived as the document's snapshot. Paths that produce one solid arrow line, such as P2 in period 1 and P3 in period 3, represent documents that have been deleted. Those documents must be archived.

A path in HP_i may be repeated in HP_j, just as P3 is repeated in HP_2 and HP_3. To make a historical link unique, the time the path was published is added to the path. Hence P3 in HP_2 is identified as P3 + T_1 and P3 in HP_3 is identified as P3 + T_2. Similarly, a current path may duplicate a path in the historical links, as is the case of P1. By adding its published time, P1 + T_3 is distinguishable from the historical link P1 + T_0. The historical links of a web site before T_i are the union of sets HP_0 to HP_{i-1}, (HP_0 U HP_1 U … U HP_{i-1}). Assuming T_i is the last time the web site has changed, then the web site's historical links are the unions of sets HP_0 to HP_i, and the current links are VP_i. The union of historical links and the current links represents all the paths that

are currently in use or have been used in the past. Every link in the union is uniquely identified by the value (path + path published time). This composite value is an example of a Temporal URL (discussed in the next section). For any path Px published at time T, the temporal URL (Px + T) uniquely identifies a document that is currently, or was once, associated with Px at time T.

3.2 Solution Designs

This section explains the scheme to track the historical links of a web site. The scheme has two major components: logging the changes to web documents and archiving deleted documents including document snapshots. The log, named TemporalURLLog, is designed to keep the history of changes to web documents. It has four fields: URL, PublishDate, ExpireDate, and NewURL. The URL field records a document's path; the PublishDate records the time the document is published; the ExpireDate field records the time this URL becomes invalid; should the change create a new URL for the document, the NewURL field records the document's new URL and serves as a link to chain all log entries related to the same document together. This log has a composite key of URL + PublishDate. The value of URL + PublishDate uniquely identifies a document in a web site's history. The Archive is a directory that stores deleted files and document snapshots. Those archived files are saved in the Archive using URL + PublishDate as its file name. The log is maintained according to the log maintenance algorithm described below:

TemporalURLLog maintenance algorithm Changes to web documents are recorded in the log by the following rules:

New document: When a new document is added to the web site, a new entry is entered with its path and the time the document is published. The ExpireDate and the NewURL are set to null. Hence, log entries with a null ExpireDate are current document entries. Initially, all current documents have an entry in the log with null ExpireDate and null NewURL.

Deleted document: The log entry associated with the document is the entry with the URL equal to the document's URL with a null ExpireDate. First, it locates the entry and changes its ExpireDate to the time the document is deleted. Then, it saves the deleted document in the Archive with URL + PublishDate as its file name.

Modified document: When a document is modified, its old version becomes a snapshot and its new version is treated as a new document. The log entry associated with the document is the entry with the URL equal to the document's URL with a null ExpireDate. First, it locates the entry and changes its ExpireDate to the time the document is modified. Then, it saves the old version in the Archive with URL + PublishDate as file name. Then, it adds a new entry with the same URL and the PublishDate is set to the time the document is modified. The ExpireDate and NewURL are set to null.

Consequently, log entries with a non-null ExpireDate and a null NewURL may be related to snapshots or deleted documents. For such an entry, if there exists another entry with the same URL and its PublishDate equals to this entry's ExpireDate then this entry is a snapshot entry and the snapshot it associates with can be retrieved from the Archive using URL + PublishDate as its file name. The snapshot is valid from the PublishDate to the ExpireDate. Otherwise, the entry is associated with a deleted document which can be retrieved from the Archive using URL + PublishDate as its file name.

Relocated or renamed page: When a document is relocated or renamed, its old URL is expired and a new URL is created. The log entry associated with the document is an entry with the URL equal to the document's URL and with a null ExpireDate. First, it locates the entry and changes its ExpireDate to the time the document is relocated or renamed and changes its NewURL field to the document's new URL. Then, it adds a new entry with the new URL and the PublishDate is set to the time the document is relocated or renamed. Hence, log entries with a non-null NewURL field help chain a document's log entries.

Using this log maintenance algorithm for the changes described in figure 2, the contents of the TemporalURLLog are shown in Figure 3, with five files in the Archive: $P2T_0$, $P3T_0$, $P3T_2$, $P5T_1$, and $P3T_3$. With the TemporalURLLog, a web server is able to determine that:

. A URL P2 valid between T0 and T1 is deleted, and the document it pointed to is in the Archive with the name $P2T_0$.

. A URL P3 has been modified repeatedly and is eventually deleted. All documents associated with P3 can be found in the Archive.

. An old URL P5 is now renamed to P7. It has been modified on T_3, and a copy of its snapshot can be found in the Archive with the name $P5T_1$.

. The log is able to determine that a historical link P1 is now renamed to P8.

. A URL P12 has never existed in the web site.

URL	PublishDate	ExpireDate	NewURL
P1	T_0	T_1	P4
P2	T_0	T_2	Null
P3	T_0	T_2	Null
P4	T_1	T_4	P8
P5	T_1	T_3	Null
P3	T_2	T_3	Null
P3	T_3	T_4	Null

P5	T_3	T_4	P7
P6	T_3	Null	Null
P1	T_3	Null	Null
P8	T_4	Null	Null
P7	T_4	Null	Null

Figure 3: The contents of TemporalURLLog based on changes described in Figure 2.

3.3 Snapshot-retrieving algorithm

This section presents an algorithm for retrieving document snapshots. The input to this algorithm is a URL Px with a snaptime T. It assumes Px is a current URL to simulate a scenario in which users who are viewing a current document would like to view the document's snapshot at time T. The algorithm is presented in [1] and is explained below.

It processes log entries backward starting from a document's current entry to trace back its changes in order to locate the snapshot. If the current URL's PublishDate is less than T, then the current document itself is its snapshot at T. Otherwise the backward search starts. To trace back, an entry's predecessor has one of the following properties:

1. Its URL equals the current entry's URL and its ExpireDate equals the current entry's PublishDate. This indicates the predecessor is an old version of the document and it is the document's snapshot from the predecessor's PublishDate to its ExpireDate.
2. Its NewURL equals the current entry's URL and the ExpireDate equals the current entry's PublishDate. This indicates the predecessor has been renamed or relocated.

Initially, the current document is treated as a candidate for the snapshot. Before reaching past time T, whenever an old version is found it becomes the new candidate for the snapshot. Note that an old document's life may span before its PublishDate. If a document Y is originally derived from a document X through a series of renaming or relocation, then this document's life span is from document X's PublishDate to document Y's ExpireDate. The search reaches the last entry when its PublishDate is before T or it no longer has a predecessor. If the last entry's PublishDate is greater than T then snapshot does not exist at T.

To illustrate, if a request P7 with snaptime T_2 is submitted, the backward search will pick up three entries: (P7, T_4, Null, Null), (P5, T_3, T_4, P7), and (P5, T_1, T_3, Null). They show that the document associated with P7 has been unchanged since T_3 and was originally associated with P5 and renamed to P7 at T_4. The document has been modified at T_3. Therefore, the algorithm returns Archive(P5 + T_1) as a snapshot. If P7 with snaptime T_3 is submitted, the backward search will pick up two entries: (P7, T_4, Null, Null)and (P5, T_3, T_4, P7) and return Document(P7) as a snapshot.

If a request P8 with snaptime T_0 is submitted, the backward search will pick up three entries: (P8, T_4, Null, Null), (P4, T_1, T_4, P8) and (P1, T_0, T_1, P4). These show that the document associated with P8 has been renamed at T_1 and T_4. It has not been modified since T_0. The algorithm will return Document(P8) as a snapshot.

4. WEB DOCUMENT SNAPSHOTS DELIVERY METHODS

This section discusses schemes for users to submit a request for a snapshot. Such requests must include the document's URL and snaptime required by the snapshot-retrieving algorithm. Three schemes for submitting a request are proposed below:

Using TemporalURL A temporal URL is a URL submitted with temporal requirements of which the documents associated with the URL must meet [1]. A typical way to submit additional information with a URL is through query strings. A query string is a set of name=value pairs appended to a URL. It is created by adding a question mark (?) immediately after a URL followed by name=value pairs separated by ampersands (&). For instance, a Snaptime query string may be defined as: Px?Snaptime=T to retrieve the web document's snapshsot at time T.

Problems with using TemporalURL are: (1) implementing this scheme requires the web server be changed to distinguish between requests for regular documents and requests for snapshots. This causes interruption for non-snapshot requests. (2) Users must be informed of such service and be trained to use temporal URL.

Using a snapshot management site This is a site designed to handle snapshot requests. Its objectives are: (1) educating users about the web document snapshot systems and (2) providing interface to enter request for snapshots. The interface can be a form with textboxes to enter web document's URL and snaptime. The snapshot-retrieving algorithm can be implemented with the server-side program that handles the form.

The advantage of this approach is that the web server does not need to change its behavior because of snapshots.

Using web services A web service is an application logic accessible via the Internet [9]. It is defined as a function that takes inputs from its arguments to perform a task. It uses standard Internet protocols such as HTTP and SOAP for communication. Inputs to a web service can be submitted with HTTP GET, or HTTP POST, or enclosed in a SOAP Envelop encoded in XML. The output from the web service is typically returned in a SOAP Envelop encoded in XML. The inputs and outputs are all in text format. A web service can be

accessed from a browser. In this mode, it will provide an interface page with textboxes to enter inputs to the service along with an information page explaining input data type. It can also be integrated in an application where it is referenced as a component. In this mode, it is used as a function where the application provides inputs through its arguments.

The snapshot-retrieving algorithm can be implemented as a web service that takes URL and snaptime as its inputs and returns the document snapshot as output. Because web services communicate in text, the snapshot returns to the user will be in text format. This implies that it is the web document's code that is sent to the user, not the web page as displayed with a browser. Hence any server-side program in the web document is unable to run. Therefore, it is a level 1 snapshot.

5. SUMMARY

Web document snapshots are useful for archiving or performing business analysis with historical data. This paper defined four levels of snapshot contents where the level 1 and level 2 snapshots involve resources managed by the web site, and level 3 and level 4 involve resources not managed by the web site. This paper proposes a web document snapshot management system designed to deliver level 2 snapshots. The system consists of a Database Snapshot Manager for providing database snapshots and a Web Document Snapshot Manager for providing web document snapshots. Algorithms supporting the two managers are presented. These algorithms use logs with time stamps to record changes to the database and web documents. With the logs, snapshots can be generated dynamically when requested. Three types of interface between the web site and users regarding snapshot delivery are proposed: temporal URL, snapshot handling site, and web service. With the proposed snapshot management system, a web site is able to deliver snapshots with dynamic contents.

REFERENCES

[1] Chao, D. (2003). Tracking a Web Site's Historical Links with Temporal URLs. Proceedings of the 3rd International Conference on Electronic Business, Singapore, 2003.

[2] Chao, D., Diehr, G., & Saharia, A. (1996) Maintaining Join-based Remote Snapshots Using Relevant Logging. Proceedings of the Workshop on Materialized Views, ACM SIGMOD, Montreal, Canada, 1996.

[3] 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.

[4] Labrinidis, A. & Roussopoulos, N. (2000). Webview Materialization. ACM SIGMOD International Conference on Management of Data, May 14-19, 2000.

[5] 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.

[6] Smithsonia Institution, "Archiving Smithsonia Websites", http://www.si.edu/archives/archives/websitepilot.html#intro (May 2003)

[7] U.S. National Archives & Record Administration, "Federal Web Site Snapshot Information", http://www.archives.gov/records_management/web_site_snapshot/snapshot.html,

[8] Utah State Government, "Draft Electronic Records Policy", http://www.archives.state.ut.us/recmanag/electronicpolicy.htm, Feb. 2002

[9] W3C World Wide Web Consortium, "Web Services Architecture", http://www.w3.org/TR/ws-arch, Feb. 2004