Towards a Temporal World-wide Web : A Transaction-time Server

Curtis E. Dyreson
School of Electrical Engineering and Computer Science
Washington State University
Pullman, Washington
E-mail: cdyreson@eecs.wsu.edu
URL: http://www.eecs.wsu.edu/˜cdyreson

Abstract

Transaction time is the time of a database transaction, i.e., an insertion, update, or deletion. A transaction-time database stores the transaction-time history of a database and supports transaction timeslice queries that retrieve past database states. This paper introduces transaction time to the World-wide Web. In a web context, transaction time is the modification time of a resource such as an XML document. A transaction-time web server archives resource versions and supports transaction timeslice. Unlike a database server, a web server is typically uninvolved in the update of a resource, instead it is only active when a resource is requested. This paper describes a lazy update protocol that enables a web server to manage resource versions during resource reads. An important benefit of our approach is that transaction-time can be supported by a transparent, minimal web server extension; no changes to legacy resources, HTTP, XML, or HTML are required. Furthermore, a web server can seamlessly become a transaction-time sever at any time without affecting or modifying the resources it services or other web servers.

[**Keywords**: web, database, servers, transaction time]

1. Introduction

Over the past two decades there has been a substantial amount of research on extending databases to support time [7, 11, 10, 4, 12]. This research has led to the development of *transaction-time databases* [8, 6, 9]. Transaction time is the time when a particular fact is stored in a database and considered current, i.e., the time between when it is inserted and deleted (an update is modelled as a deletion followed by an insertion). Very briefly, a transaction-time database stores all of the past states of a database and allows queries, called transaction timeslice, to retrieve any desired past state.

This paper applies transaction-time database technology to the web. An HTTP server (also called a web server) is like a database server insofar as it services a large body of information. Consequently it is unsurprising that database techniques can be used to enhance the functionality of web servers. Each time a resource on the web is edited a new *version* of a resource is created. The transaction-time of a web resource is the time when the version is considered current, that is, it is the time between when that resource was created and deleted.

A transaction-time web server offers two important benefits to users. First, it supports transaction timeslice. Transaction timeslice is a query that retrieves a web resource as of a specified transaction time. Such queries are important in many situations, such as those listed below.

- A legal dispute involves a page (since deleted) that advertised a Kylie Minogue CD for $2.95 on 21-Jan-2000.
- A cricket fan desires to relive the Australian team's success in the Ashes by rereading The Australian as of 27-Dec-1997.
- A late tax payer wishes to obtain a taxation form for 1998.
- A student wants to obtain the homework solutions from the previous year's version of a database subject. (We also discuss strategies whereby a lecturer can expire those solutions to prevent future student access.)

- Users complain about the redesign of a page, so management decides to rollback to the previous version.

A second advantage of a transaction-time web server is that it solves (in part) the *broken link* problem. A broken link is either a link to a resource that no longer exists or a link to a resource that has been modified to such an extent that it no longer contains relevant information. The number of such links at a site varies, estimates run to as high as 30% although 5% to 10% is more likely [5]. Search engines often have many broken or irrelevant links since search engines usually visit individual sites periodically, say, every few months, but sites change more rapidly. The root cause of the broken link problem is that resources evolve *independently*, but have *(time) dependent* hyperlinks. In a very real sense, every hyperlink is a transaction-time query: a request for information that was relevant as of the time that the link was created.

The target audience of this paper is a person with knowledge of databases and the web. We assume familiarity with common web features such as browsers, servers, and Uniform Resource Locators (URLs), and common database concepts such as SQL queries and tables.

2. Motivating Example

Each day Sally reads the sports section of an on-line Australian newspaper, available from the following (fictional) URL.

```
tabloid.au/sports.htm
```

Sally is feeling depressed. She thinks back to Cathy Freeman's victory in the Women's 400m at the Sydney Olympics on 25-Sep-2000 and desires to read about her race again. In particular Sally wants to read the sports section of the on-line paper *as of* 26-Sep-2000 the day after the race. Sally has two hopes.

Sally can go to the paper's website and hunt for an *archive* of past news articles. To access the archive (assuming that it exists) Sally uses a substantially different navigation technique than when she accesses the current sports section. While Sally can go directly to the current sports section using the URL given above, in order to use the archive, Sally will have to first find it, most likely, by following a link from the newspaper's home page. Once the archive is reached then another link must be followed to the day in question, and perhaps from there to the sports page. A different, but common interface to an archive is a form that permits individual articles to be retrieved using a keyword search. When Sally reaches the archived article, she finds some of the links and images are broken. For instance there is a link to an article about the Cathy at `www.espn.com` which no longer works. In addition, an image advertising Duff Beer is absent (although Sally regards this in a positive light).

Alternatively, Sally can retrieve the sports section *exactly* as it looked on 26-Sep-2000 by using the following URL.

```
tabloid.au/sports.htm?26-Sep-2000
```

All images are retained. Links to other (transaction-time) servers still work, exactly as they did on that day.[1].

The second option is what can be offered by a network of transaction-time web servers. A transaction-time web server stores the history of each resource that it services, and provides a URL-based query mechanism by which past versions can be easily accessed. When retrieving a past version of a resource the state is retrieved *in toto*, including links to other resources as of the archived time.

3. Time Model

Research in temporal databases has identified two primary, distinct time dimensions: *valid time* and *transaction time* [3]. Valid time is the real-world time of a fact, whereas transaction time is the database time when that fact was present in the database. For instance, the valid time of a fact that describes the race run on 25-Sep-2000 is 25-Sep-2000. Assuming that fact was inserted on 26-Sep-2000 and deleted on 30-Sep-2000, the transaction time is the temporal interval [26-Sep-2000, 30-Sep-2000). Transaction time is typically represented as an interval. In the case of a page or image, the transaction time is the time between edits. Each edit creates a new version.

The transaction-time domain is a set of instants, $T_{tt} = \{0, \ldots, \infty\}$ where 0 represents the earliest web server time and ∞ is the latest time. The earliest time will vary from server to server, depending upon when the transaction-time service is enabled. In this paper, example times will be represented using Gregorian calendar conventions in the granularity of days, so each instant in the transaction-time domain corresponds to a day. In practice, the representation for and granularity of the

[1]XLink offers a much richer medium for adding transaction time to links [13], but the recommendation is not yet supported by many products.

transaction-time domain is server dependent, with a granularity of UTC seconds and a Gregorian calendar representation being the most common. In contrast to temporal databases, the transaction-time domain for web servers includes *future* transaction times. This enables page authors to set expiration limits on pages and to schedule pages for future release.

4. A Transaction-time Server

This section describes the issues involved in extending a web server to support transaction time. There are three main issues. The first issue is specifying the additional functionality that a transaction-time web server should support. The second issue is determining the transaction-time lifetime of each *version* of a document. This issue is complicated by the fact that documents often are composed of or depend upon several files. The final issue is how to engineer a web server to maintain resource versions and to respond to transaction-time queries.

This section does not address *dynamic* resources. A dynamic resource is a resource generated as the result of a program evaluated by the web server. Common dynamic resources are Active Server Pages (ASPs), pages generated by Common Gateway Interface (CGI) scripts, and pages produced at run-time from back-end databases. For simplicity this paper focuses on static resources, that is, those which a server obtains from a local file system. Dynamic resources are more difficult to archive since both the resource and the processing environment active at the time must be archived (e.g., the server and server extensions). When a past version is retrieved it is dynamically regenerated using the past processing environment.

4.1. A Web Server Model

Definition 4.1 [Web Server]
A web server, $\mathcal{W}$, can be abstractly modelled as a function that maps a URL, u, to a response, (c, r), where c is the response code and r is the resource, i.e., $\mathcal{W}(u) = (c, r)$. ▮

$\mathcal{W}$ is a total function since a server will always return some response, e.g., a request for a deleted resource will receive a 404 code along with a page indicating that the resource is not found.

Definition 4.2 [Transaction-time Web Server]
A transaction-time web server, $\mathcal{W}_{tt}$, is a function that maps a transaction-time URL, (u, q_{tt}, p_{tt}), where u is a URL, q_{tt} is a transaction-time query, and p_{tt} is a restructuring query, to a response, i.e., $\mathcal{W}_{tt}(u, q_{tt}, p_{tt}) = (c, r)$. ▮

The transaction-time query, q_{tt}, is built using the following BNF rules.

$$\begin{array}{lcl}
\langle \mathit{query} \rangle & ::= & \epsilon \mid [\langle \mathit{sequence} \rangle]\, \langle \mathit{time} \rangle \\
\langle \mathit{time} \rangle & ::= & \text{time literal} \mid \texttt{timeOf} \\
\langle \mathit{sequence} \rangle & ::= & [\langle \mathit{sequence} \rangle\ \texttt{.}]\,(\texttt{pred} \mid \texttt{succ})
\end{array}$$

For syntactic convenience, if the time is missing then the current time, `now`, is used. The predecessor (successor) operation retrieves a prior (later) version of a resource, e.g., `pred.pred.Sep-26-2000` would retrieve the version prior to the previous version. The `timeOf` function retrieves the time of the document that is being retrieved (useful to determine the time of the predecessor or successor).

The restructuring query, p_{tt}, is used to rewrite the hyperlinks within the resource during retrieval, that is, it becomes the transaction-time query on links emanating from the retrieved resource. It has the same syntax as the transaction-time query. Several examples are given below to demonstrate the utility of separating querying from restructuring.

- Retrieve the current version of u and links from the current version.

  $\mathcal{W}_{tt}(u, \texttt{now}, \texttt{now})$

  This is the default for non-transaction-time queries.

- Retrieve the version of u as of Sep-26-2000 with links to other documents as of that time.

  $\mathcal{W}_{tt}(u, \texttt{Sep-26-2000}, \texttt{Sep-26-2000})$

  Links are rewritten to retrieve past document versions.

- Resurrect the version of u as of Sep-26-2000 as though it were a current document.

  $\mathcal{W}_{tt}(u, \texttt{Sep-26-2000}, \texttt{now})$

  Links in u are to current documents.

- Retrieve the predecessor of u as of the time the version was created.

  $\mathcal{W}_{tt}(u, \texttt{pred}, \texttt{timeOf})$

Links in the predecessor of u are to documents current as of the time the predecessor was created.

- Retrieve a sequence of predecessors of u.

  $\mathcal{W}_{tt}(u, \texttt{pred}, \texttt{pred})$

  Links in the predecessor of u are to predecessors.

4.2. Encoding the Transaction Time Query in a URL

A network of transaction time web servers is possible only if there is a standard convention for specifying transaction-time web server operations. The current scheme for URL specification does not include transaction time. One problem is that humans specify times in a variety of ways. There are many different calendars, in many different languages currently in use. Dates in one calendar, say the Islamic calendar have little in common with dates in the Gregorian calendar. Second, dates and times within a single calendar have many different interpretations. A common example is the reversal of day and month fields between the United States and the rest of the world. Third, date literals can be expressed in a wide variety of formats, e.g., YYYYMMDD vs. DD-MON-YYYY. Fourth, times are sometimes given symbolically, e.g., Palm Sunday, 2000. Fifth, times are often specified to a coarse granularity, e.g., 2000. Given the richness of language for time, encoding transaction-time within a URL necessarily involves extreme simplification.

Our strategy is very simple: the transaction-time query and restructuring operation are appended to the resource as *queries* [1] using '?' followed by the respective queries separated by a comma. The advantage of this scheme is that requests to non-transaction-time servers will function exactly as before since the query portion is ignored in HTTP requests for static resources (queries for dynamic resources are not-included in this strategy). The following example URLs illustrate the strategy.

- Retrieve the current version of `sports.html`.

  ```
  sports.html?now,now
  ```

  Since the default timeslice and restructuring queries are `now` the following URL has the same effect.

  ```
  sports.html
  ```

  Hence, the scheme is completely backwards-compatible with existing URLs.

- Resurrect the previous version of `sports.html` as though it were the current version.

  ```
  sports.html?pred,now
  ```

  A non-transaction-time web server ignores the query for static documents, so it will result in the fetch of the current resource. A transaction-time web server will fetch the predecessor, but will not restructure links in the predecessor.

- Retrieve the version as of `26-Sep-2000`.

  ```
  sports.html?26-Sep-2000,26-Sep-2000
  ```

XLink offers a much richer medium for adding transaction time to links [13]. The XLink recommendation does not include transaction time, although a transaction time element may be added to the link's content with impunity.

4.3. Resource Versions

$\mathcal{W}_{tt}$ associates a transaction-time interval with each resource to record the lifetime of that resource. We assume that $\mathcal{W}_{tt}$ maintains a table, called the resource history table, R_H. Each $v \in R_H$ is a tuple (u, r, b, e) where u is the URL for the resource, r is a pointer to the resource which is stored elsewhere (see Section 4.4 for a discussion of storage alternatives), b is the transaction start time, and e is the transaction stop time. The transaction time of the association between the URL, u, and the resource, r, is the interval of time $[b, e)$. The transaction time of the association is not quite the same as the lifetime of this *version* of the resource. Resource versions have to be tracked since users can retrieve the predecessors and successors of resources from a transaction-time web server. The difficulty is that each resource might depend on other resources. We stipulate that a new version is created each time a resource or one of its dependents is modified. A common dependency is an image. When a resource is retrieved, the images that it depends upon can be included or excluded.

If a resource has no dependencies then the predecessor and successor of a version is well-defined and directly related to the transaction time. The predecessor of a version v is x such that $x \in R_H \wedge x.e = v.b$ while the successor of v is w such that $w \in R_H \wedge v.e = w.b$. A resource version need not have a successor or predecessor.

If dependents are included, however, identifying versions is more difficult. Basically, a new version is created whenever a resource or one of its dependents is updated. The predecessor of a version v_0 with included dependents $v_1, \ldots, v_n$ is x_0 with included dependents $x_1, \ldots, x_n$ such that $x_i = v_i$ unless $x_i.e = max(x_0.b, x_1.b, \ldots, x_n.b)$ in which case $x_i = \text{predecessor}(v_i)$. The successor is defined similarly.

As an example, assume the following page has two included images all of which are shown with their transaction-time lifetime.

```
sports.htm,[26-Sep-2000,30-Sep-2000)
duff.jpg,  [1-Aug-2000,30-Sep-2000)
cathy.gif, [28-Sep-2000,30-Sep-2000)
```

If the transaction-time query for `sports.htm` is

```
pred.26-Sep-2000.
```

and images are included, then the predecessor of `cathy.gif` is used. If the query is

```
pred.pred.25-Sep-2000.
```

then the predecessor of `sports.htm` is used (along with that of `cathy.gif`.

4.4. Resource Storage

Software packages for the efficient storage and retrieval of resource (file) versions are commonplace, e.g., diff files and RCS. Any of these packages can be used by a transaction-time web server. In the resource history table, each tuple contains a ‘pointer’ to a version. We assume that this pointer has all the necessary information for retrieving the appropriate version from the version storage system. A full exploration of the tradeoffs among the various packages is beyond the scope of this paper.

4.5. A Lazy Transaction Protocol for Static Resources

In this section we develop a transaction protocol that maintains the resource history table, R_H.

An important difference between a database server (in a client/server model) and a web server is the lack of an update protocol. In a database a user has to execute specific instructions to modify data. For instance in SQL a user must execute an `UPDATE TABLE` statement to modify the information in a table. Consequently, a database can identify when data is being changed. In contrast a web server is usually uninvolved when a resource author modifies a web resource. Resource authors can edit resources directly (e.g., using emacs), generate resources using a filter (e.g., latex2html) or generate resources from a database. None of these resource creation alternatives requires a web server. As long as the resource is located where the server can read it (in a suitable directory with the proper protections) the resource is made available from the server.

The remaining task is to engineer the web server to archive past versions of resource during a read of the resource. Figure 1 is the lazy transaction protocol. The basic idea is to update resource history table while responding to the (transaction-time) HTTP request. The resource history table must be updated if the the file modification time of a requested resource is different from the information stored in R_H, the resource has been deleted, or the resource has just been created. Each of these actions creates a new version of the resource. Note that q_{tt} and p_{tt} default to the current transaction time, `now`. Recall that $v \in R_H$ is a tuple (u, r, b, e) where u is the URL for the resource, r is a pointer to the resource, b is the transaction start time, and e is the transaction stop time.

One limitation of this strategy is that a web server will miss resource versions that are never read. So if a version is created and then deleted without being read, the server will not archive that version. In effect, the version did not exist for the server. Resource authors can force a version to be archived by reading it from the web server. This is already common practice since authors check correctness of an edit by examining the result with a browser. Another strategy to minimise the risk of missing a resource version is to periodically run a web robot to request all the resources from a site and thereby force the archiving of resource versions.

The lazy protocol incurs an additional overhead on each HTTP request: a lookup in R_H. The cost of this step can be minimised by either indexing R_H appropriately (on both the URL and transaction time) or pinning the set of current URLs in memory to avoid the lookup. Additionally, if a resource has been modified, a database update and insert may be needed.

4.6. Improving Transaction Timeslice Behaviour

The information in the resource history table can be utilised and modified to improve responses to requests for resources. Consider the following scenarios.

- *Resource renaming* - Often a resource is moved or renamed within a server file system. If a resource

```
subroutine HTTPRequest(URL $u'$, Query $q_{tt}$, Query $p_{tt}$)
  -- Does the resource exist at the queried time?
  $R$ := SELECT $r$ FROM $R_H$
          WHERE $u' = u$ AND $q_{tt}(b, e)$
  -- If found return the restructured resource
  if $R$ is not empty then
      return HTTPRespond(200, $p_{tt}(R)$)

  -- Was it a request for an invalid version?
  if $q_{tt}$ != now then
      return HTTPRespond(404, 'Version not found')

  -- Fetch the file, $F$ corresponding to the URL
  $F$ := OperatingSystemReadFile($u'$)
  if ReadFailed then
      -- Update the lifetime of a deleted $u'$
      UPDATE $R_H$ SET $e$ = current
          WHERE $u = u'$ AND now IN $[b, e)$
      -- Commit the DB changes
      COMMIT
      -- Send the error to the HTTP Client
      return HTTPRespond(404, 'Page not found')

  -- Determine when the file was last modified, $T$
  $t$ := LastModifed($F$)

  -- Determine whether the URL, $u'$ is current
  $R$ := SELECT *
      FROM $R_H$
      WHERE $u' = u$ AND now IN $[b, e)$

  if $R$ is empty then
      -- $u'$ has yet to be archived, insert a new version
      INSERT INTO $R_H$ VALUES ($u'$, $F$, $t$, $\infty$)

  -- Update database for a recently modified resource
  if $R.b < t$ then
      -- Terminate the lifetime of the current version
      UPDATE $R_H$ SET $e = t$
          WHERE $u' = u$ AND now IN $[b, e)$
      -- Insert a new version
      INSERT INTO $R_H$ VALUES ($u'$, $F$, $t$, $\infty$)
      -- Commit the DB changes

  -- Send the file to the HTTP Client
  COMMIT
  return HTTPRespond(200, $p_{tt}(F)$)
```

Figure 1. A lazy transaction-time web server request protocol

identified by the URL u is renamed to the URL u' then the history of resource u can be copied to create a history for resource u'. Transaction timeslice requests for both u and u' will now have the same behaviour.

- *Resource expiration* - The stop transaction times on a resource can be fixed to expire at a (future) time. When the current time advances past the stop time, the resource will no longer be current as of now.
- *Resource extinction* - Some resource need to be permanently removed, e.g., homework solutions from a previous year's subject. To force the extinction of a resource, the history of that resource can be deleted from the resource history table.
- *Fixing 404 responses* - A request for a deleted (or future) resource will result in a 404 response, meaning that the resource is not found. A better strategy is to automatically forward the request to the predecessor (or successor) of the resource. The forwarding page should state that the requested version does not exist and should include a link to an earlier (later) version. Hence, broken links to deleted resources can be handled quite easily by a transaction time web server.

5. Summary

This paper proposes a strategy whereby individual web servers can be seamlessly migrated to a transaction-time web. One benefit of transaction-time support is that transaction timeslice, versioning, and audit queries are supported. Another benefit is that the problem of broken links can be mitigated since the most recent version of a resource can be returned when a deleted resource is requested. Yet another benefit is better support for XLink. A non-mandatory goal in the XLink requirements specification is that "it must be possible to detect when a resource a link points to is invalidated or modified" [13]. A transaction-time web server can meet this goal The key problem in providing transaction-time support is that web servers lack the ability to track and maintain resource versions. In this paper we proposed a lazy protocol whereby any server can be easily modified to provide support for transaction time and a simple scheme for querying resource versions.

6. Acknowledgements

We would like to thank Tom Dalling for working with us on an Honour's project in implementing a transaction-time web server using Jigsaw [2]. We would also like to thank the School of Information Technology at Bond University, Gold Coast, Queensland where we worked on this paper.

References

[1] T. Berners-Lee, R. Fielding, and L. Masinter. Rfc 2396: Uniform resource identifiers (uris): Generic syntax. http://www.ics.ucs.edu/pub/ietf/uri/rfc2396.txt, August 1998.

[2] T. Dalling. Versioning of web resources. Honour's Thesis, James Cook University, November 1998.

[3] C. Jensen and C. e. Dyreson. *A Consensus Glossary of Temporal Database Concepts - February 1998 Version*, pages 367–405. Springer-Verlag, 1998.

[4] N. Kline. An Update of the Temporal Database Bibliography. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 22(4):66–80, Dec. 1993.

[5] S. a. L. G. Lawrence. Accesibility and Distribution of Information of the Web. *Nature*, 400:107–109, 1999.

[6] D. Lomet and B. Salzberg. *Transaction-Time Databases*, chapter 16, pages 388–417. Benjamin/Cummings, 1993.

[7] E. McKenzie. Bibliography: Temporal Databases. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 15(4):40–52, December 1986.

[8] E. McKenzie and R. Snodgrass. Extending the Relational Algebra to Support Transaction Time. In U. Dayal and I. Traiger, editors, *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 467–478, San Francisco, CA, May 1987. Association for Computing Machinery.

[9] J. F. Roddick and R. T. Snodgrass. Transaction Time Support. In R. T. Snodgrass, editor, *The TSQL2 Temporal Query Language*, chapter 17, pages 319–325. Kluwer Academic Publishers, 1995.

[10] M. D. Soo. Bibliography on Temporal Databases. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 20(1):14–23, Mar. 1991.

[11] R. Stam and R. T. Snodgrass. A Bibliography on Temporal Databases. *Database Engineering*, 7(4):231–239, Dec 1988.

[12] V. J. a. A. K. Tsotras. Temporal Database Bibliography Update. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 25(1):41–63, March 1996.

[13] W3C. Xml linking language (xlink) version 1.0. http://www.w3.org/TR/xlink, July 2000.