The V2 Temporal Document Database System

Kjetil Nørvåg
Department of Computer and Information Science
Norwegian University of Science and Technology
7491 Trondheim, Norway
Kjetil.Norvag@idi.ntnu.no

ABSTRACT

In this paper we present an overview of the V2 temporal document database system, which supports storage, retrieval, and querying of temporal documents.

1. INTRODUCTION AND MOTIVATION

One of the advantages of XML is that the document itself contains information that is normally associated with a schema. This makes it possible to do more precise queries, compared to what has been previously possible with unstructured data. It also has advantages for long-term storage of data: even though the schema has changed, the data itself can contain sufficient information about the contents, so that meaningful queries can be applied to the data. This is of increasing importance as storage costs are rapidly decreasing and it feasible to store larger amounts of data in databases, including previous versions of data. If documents as well as DTDs/schemas are stored in a suitable temporal database system, it is also possible to make queries based on the actual DTD/schema that was valid at the time a particular document was stored. In order to efficiently manage the temporal versions, a temporal document database system should be employed. In this paper, we describe an approach to temporal document storage, which we have implemented in the V2 temporal document database system. Important topics include temporal document query processing, and control over what is temporal, how many versions, vacuuming etc., something that is necessary for practical use in order to be able to control the storage requirements.

We have previously in the TeXOR project studied the realization of a temporal XML database using a *stratum* approach, in which a layer converts temporal query language statements into conventional statements, executed by an underlying commercial object-relational database system [3]. We could in this way rapidly make a prototype that supported storage and querying of temporal XML documents. However, we consider a stratum approach only as a short-term solution. As a result of using a stratum approach, some problems and bottlenecks are inevitable. For example, no efficient time index for our purpose were available, and query optimization can be a problem when part of the "knowledge" is outside the database system. Retrieving data from the database and process it in the TeXOR middleware would not be a good idea if we want to benefit from the XML query features supported by the object-relational database management system.

The TeXOR project demonstrated the usefulness of a temporal XML databases in general, and gave us experience from actual use of such systems. The next step is using an *integrated* approach, in which the internal modules of a database management system are modified or extended to support time-varying data. This is the topic of this paper, which describes V2, a temporal document database system. In V2, previous versions of data are kept, and it is possible to search in the historical (old) versions, retrieve documents that was valid at a certain time, query changes to documents, etc.

Although we believe temporal databases should be based on the integrated approach, we do not think using special-purpose temporal databases is the solution. Rather, we want the temporal features integrated into existing general database systems. In order to make this possible, the techniques used to support temporal features should be compatible with existing architectures. As a result, we put emphasis on techniques that can easily be integrated into existing architectures, preferably using existing index structures (history tells us that even though a large amount of "exotic" index structures have been proposed for various purposes, database companies are very reluctant to make their systems more complicated by incorporating these into their systems, and still mostly support the "traditional" structures, like B-trees, hash files, etc.) as well as a query processing philosophy compatible with existing architectures.

2. DATA AND TIME MODEL

A document version stored in V2 is uniquely identified by a *version identifier* (VID). The VID of a version is persistent and never reused, similar to the object identifier in an object database.

The aspect of time in V2 is *transaction time*, i.e., a document is stored in the database at some point in time, and after it is stored, it is *current* until logically deleted or updated. We call the non-current versions *historical versions*. When a document is deleted, a tombstone version is written to denote the logical delete operation.

The time model in V2 is a linear time model, and V2 also supports reincarnation, i.e., a (logically) deleted version can be updated, thus creating a non-contiguous lifespan, with possibility of more than one tombstone for each document. Support for reincarnation is particularly interesting in a document database system because even though a document is deleted, a new document with the same name can be created at a later time (in the case of a web data warehouse this could also be the result of a server or service being temporarily unavailable, but then reappear later).

3. FUNCTIONALITY

V2 provides support for storing, retrieving, and querying temporal documents. It is possible to retrieve a document stored at a particular time t, that was valid in a certain time period p, etc. V2 also supports some basic operators. In contrast to many existing systems that support versioning of documents, time is an integrated concept of V2, and is efficiently supported by the query operators.

WWW2003, May 20–24, 2003, Budapest, Hungary.
ACM xxx.

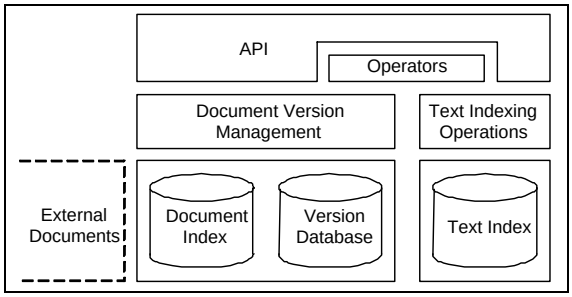

Figure 1: The V2 prototype architecture.

Items in temporal and document databases have associated metadata that can be used during query processing to filter out a subset of items for subsequent query processing (this metadata would normally be stored as ordinary data in the tuples/objects in relational/object databases). The result of execution of the operators (i.e., VIDs, document names, timestamps, periods, etc.), can either be used in subsequent "traditional" query processing employing traditional operators like selection, projection etc., or the VIDs can be used to retrieve the actual document versions from the version database. V2 supports a number of operators, including operators for returning the document identifiers of all document versions containing one or more particular words, selecting from a set of versions those versions that were valid at a particular time or during a particular periods, as well as the Allen operators, i.e., *before, after, meets*, etc.

V2 supports automatic compression of documents if desired (this typically reduces the size of the document database to only 25% of the original size). In addition, space-reduction can be achieved by traditional vacuuming (delete all non-current versions created before a certain time t) and granularity reduction (delete intermediate versions).

4. DESIGN AND IMPLEMENTATION

The current prototype is essentially a library, where accesses to a database are performed through a V2 object, using an API supporting the operations and operators described previously. The bottom layers are built upon the Berkeley DB database toolkit [4], which we employ to provide persistent storage using B-trees.

The architecture of V2 is illustrated in Figure 1, and the main modules are the version database, document name index, document version management, text index, API layer, operator layer, and optionally extra structures for improving temporal queries. These modules/structures will now be described in more detail.

Version database: The document versions are stored in the version database. In order to support retrieval of parts of documents, the documents are stored as a number of chunks (this is done transparently to the user/application) in a tree structure, where the concatenated VID and chunk number is used as the search key. The VID is essentially a counter, and given the fact that each new version to be inserted is given a higher VID than the previous versions, the document version tree index is append-only. This is interesting, because is makes it easy to retrieve all versions inserted during a certain VID interval (which can be mapped from a time interval). One application of this feature is reconnect/synchronization of mobile databases, which can retrieve all versions inserted into the database after a certain VID (last time the mobile unit was connected). In order to support reverse mapping from VID to document name and time, this information, together with other low-level metadata, is stored in a separate meta chunk together with the document.

Document name index: A document is identified by a *document name*, which can be a filename in the local case, or URL in the more general case. Conceptually, the document name index has for each document name some metadata related to all versions of the document, followed by specific information for each particular version. For each document, the document name and whether the document is temporal or not (i.e., whether previous versions should be kept when a new version of the document is inserted into the database) is stored. For each document *version*, some metadata is stored in structures called *version descriptors*: 1) timestamp and 2) whether the actual version is stored compressed or not.

Document version management: Store and retrieve documents, employing the document name index and version database.

Text indexing: A text-index module based on variants of inverted lists is used in order to efficiently support text-containment queries, i.e., queries for document versions that contain a particular word (or set of words).

In our context, we consider it necessary to support dynamic updates of the full-text index, so that all updates from a transaction are persistent as well as immediately available. This contrasts to many other systems that base the text indexing on bulk updates at regular intervals, in order to keep the average update cost lower.

In order to support temporal text-containment queries we provide several time-indexing techniques, each suitable for particular application domains (discussed in more detail in [2]), including the *TV index* which is a summary time index mapping from time to VID, and the *VP index* which indexes validity time for versions, essentially mapping from VID to time period for every version.

5. CONCLUSIONS

We have in this paper described the V2 temporal document database system, which supports storage, retrieval, and querying of temporal documents. All of what has been described previously in this paper is implemented and supported by the current prototype. Such a system is useful in a number of application, and the prototype has for example been used to manage a temporal XML/web warehouse, storing the history of a set of selected web pages or web sites.

For a more detailed study of the design, implementation and performance of V2, we refer to [1].

6. REFERENCES

[1] K. Nørvåg. The design, implementation and performance evaluation of the V2 temporal document database system. Technical Report IDI 10/2002, Norwegian University of Science and Technology, 2002. Available from http://www.idi.ntnu.no/grupper/DB-grp/.

[2] K. Nørvåg. Supporting temporal text-containment queries. Technical Report IDI 11/2002, Norwegian University of Science and Technology, 2002. Available from http://www.idi.ntnu.no/grupper/DB-grp/.

[3] K. Nørvåg, M. Limstrand, and L. Myklebust. TeXOR: Temporal XML Database on an Object-Relational Database System. In *(submitted for publication)*, 2002.

[4] M. A. Olson, K. Bostic, and M. Seltzer. Berkeley DB. In *Proceedings of the FREENIX Track: 1999 USENIX Annual Technical Conference*, 1999.