Temporal XML Data Warehouses: Challenges and Solutions

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

Abstract

Increasing amounts of data is stored in XML. In order to facilitate more efficient querying on this data, and in particular on data from several sources (for example Web sites), data can be loaded into an XML data warehouse. *Often, we want to query historical document versions, or query changes between document versions. This can be facilitated by a* temporal *XML data warehouse (XML-DW). In this paper, we study some of the challenges we meet in an XML-DW, in particular those related to consistency, and propose how these challenges can be met. We also identify some problems that are difficult to solve automatically in a temporal XML-DW. Many of the issues described in this paper are also applicable in a broader context, for example for general Web warehouses storing on HTML web pages.*

1 Introduction

Increasing amounts of data is stored in XML format. This includes documents previously stored in various proprietary formats in files, as well as data previously stored in databases. The XML data is preferably stored in an "XML database management system" (XML-DB). This can be a native XML database, or one of the XML-enabled database systems. An XML-DB should support all the properties normally found in database systems (queries, recovery, concurrency control, etc.), as well as somehow providing support for new features required as a consequence of XML. This requires, for examples, new/extended query languages.

A specialized version of database systems that has evolved during the last decade, is *data warehouses*, which typically is a copy of transaction data specifically structured for querying and reporting. The data warehouse concepts are also of interest in the context of XML data. However, one difference is the external data that can be stored in an XML data warehouse. While we in a typical "traditional" data warehouse store data from one or more of the databases controlled by the company (or institution), an XML data warehouse can also contain external data, for example collected from the Web (as in the case of Xyleme [9]). The latter is often called a *Web warehouse*, but we will in this paper use the term *XML data warehouse* (XML-DW) to capture the aspect that the techniques described in this paper could be used for any type of warehouse storing XML data. However, as will be seen, the context of data from the Web introduces some new interesting challenges, especially in the context of temporal data.

A feature still only partially supported by traditional database systems, is support for management and query of temporal data. This includes managing several versions of an item, and support for queries involving versions valid at different times. We consider support for temporal data at least equally important in the context of XML-DBs. In many cases, we actually consider it a *more important*. One reason for this, is that in a traditional database system temporal aspects can trivially be integrated into schemas, and users/applications have to obey the rules of these data schemas. In contrast, much of the data/documents available is generated in a much more ad-hoc way. A typical example is various documents available on the Web. For these, we can not in a trivial way force the aspects of time into the documents.

Managing and querying data differs in many aspects between "traditional" database systems, and XML-DWs. One issue is structure (e.g., using XPath) and text-related queries. This was the topic in [7]. Another issue is the fact that management of the sites where data is stored is autonomous, and the fact that we can not retrieve the data from all sites in the same time instant. The result can be in inconsistency when documents from one site reference documents from another site. However, it is also important to notice that this consistency problem between versions also occur within one site, even between documents managed by the same person. Although the updates of various documents having inter-document relationships should ideally be done inside a transaction, this is seldom the case. Also,

it is very easy to forget to update a document referring to another document that has been updated.

The organization of the rest of this paper is as follows. In Section 2 we give an overview of related work. In Section 3 we describe some of the issues and challenges of temporal XML-DWs. In Section 4 we discuss possible solutions to these problems. Finally, in Section 5, we conclude the paper.

2 Related work

An approach for introducing valid time features into XML documents, *The Valid Web*, is presented by Grandi et al. in [4].

An application that illustrates one application area for XML warehouses, is the transaction-time web server described by Dyreson [3]. Given a certain URL for a page on the server and a given date t, the page valid at date t is returned to the client. To introduce temporal aspects into XQuery based approach, one approach is to extend XPath, as described by Dyreson in [2].

Representing Web data in a warehouse introduces problems related to partial and inconsistent information, and a data model for semistructured data with partial and inconsistent information is presented by Liu and Ling in [5]. A theoretical framework for an XML-DW with incomplete information is described by Abiteboul et al. in [1]).

In previous papers [7, 8], we have described temporal query operators for XML databases, and algorithms that can be used to execute these operators. In [7, 8], the operators are described in the context of an XML query language based on OQL. However, these approaches do not take into account missing versions and inconsistency that can exit in an XML-DW.

3 Issues and challenges

In this section, we will describe some of the issues and challenges of temporal XML-DWs.

3.1 Aspects of time in XML data warehouses

In temporal databases we have different aspects of time, where the two most common aspects are transaction time and valid time. In the context of storage of XML documents, we have two cases which from a query point of view are similar to transaction time:

- Local storage of documents (e.g., in a database system storing XML documents), where we have full information about time of creation/storage of an XML document. In this case, time is exactly transaction-time equivalent.
- XML data warehouse or other non-synchronized storage of copies of XML documents. Although similar to transaction time, this is not exactly the same. Important differences are:
  - In this case we in general do not know the time of creation/storage of an XML document, only the time when the document was retrieved from the Web ("crawled").
  - The documents in the warehouse are not retrieved at the same point in time, the result is an inconsistent view of the documents. For example, a document can have references to a document not yet retrieved, or a document that has already been deleted.
  - There might have been updates between the versions we have retrieved, i.e., we do not necessarily have all the versions of a particular document.

A third case, which have similarities to valid time, is document time. Many documents include a timestamp in the document itself. This can for example be the time the document was written, or when it was posted. Examples are news notices from the news agencies. The documents can also be indexed and queried based on this document time. Although it could be difficult to extract this time from a document automatically, we can expect many documents to include this metadata in a "standardized" way, based on RDF. One example is XMLNews-Meta (based in part on RDF), which if used can provide meta-information such as publication time and expire time.

3.2 Time approximation and inconsistency between versions

When querying a temporal XML-DW, time approximation and inconsistency between versions will be important issues. The main reasons for these problems are that management of the sites where data is stored is autonomous, and the fact that we can not retrieve the data from all sites in the same time instant. As a result, we may miss some document updates, and there can be inconsistencies when documents from one site reference documents from another site. These issues will now be studied in more detail.

3.2.1 Time approximation

An issue related to time is the assumed *content* at a particular time. Assume we have one document version from time t_1, and another version of the document from t_2 (where $t_2 > t_1$). What can we assume is the content at time t, where $t_1 < t < t_2$? Some options are:

- Assume the contents for all t where $t_1 < t \leq t_2$ is the same as the contents at time t_1.
- Assume that the document closest in time is the most appropriate, i.e., choose document version with timestamp t_i which minimizes $|t - t_i|$.
- Assume the contents is only valid for the times retrieved (with a given granularity, for example a given day), and undefined/non-existing for all other times. An example is the number of people present on a football match at the Stade de France.

The two first choices are useful. The last one is in many ways the most correct, but can be approximated by the first option if the refresh rate is sufficiently high. Sufficiently high in this case, means frequent enough to capture most document states. This can be achieved using adaptive refreshing.

Rather than solving the problem by choosing a particular version using one of the approaches above, another approach is to use an interpolated value for the contents at t. Examples where this solution could be applicable is for example the temperature in Paris, or the exchange rates for the dollar. However, using interpolated values is difficult because we need good knowledge of the semantics of documents (although this could change in the future, given certain standardized DTDs and resource information).

3.2.2 Inconsistency between versions

In an unsynchronized XML-DW (for example a web warehouse), documents will in general be inconsistent because they are retrieved at different times. As a result, there can be:

- References to documents not yet inserted into the data warehouse (not crawled yet). This inconsistency could be resolved by retrieving the document at query time, although this might not be practical in all situations, e.g., when the query involves many or large documents.
- References to contents in other documents. It is possible that the version of the referenced document we have available, is only a previous version where the actual contents that is referenced either was not yet inserted, or contents is outdated with respect to the referring document. This case could not be resolved without checking all affected documents to check that we already have the most recent version.

In addition, in a *temporal* XML-DW, there can be inconsistencies as a result of 1) references to missing versions, because the document/page was not crawled often enough, and 2) References to a document that is completely missing. The reason could be that it was removed during vacuuming (see Section 3.4). These two temporal inconsistencies can not be completely resolved, because the information is not available anymore.

It should be noted that the problems described here has similarities to general data integration problems. However, in our context it is not the data heterogeneity that creates the problem, it is the "shift in time".

3.3 Use of temporal queries in continuous queries

Continuous queries are queries that are executed regularly, for example once a week (For a more detailed treatment of continuous queries in general, we refer to [6]). Continuous queries can be categorized as follows:

1. Queries where each query only considers the contents of one document. This has similarities to monitoring queries (filtering) which are executed in an XML data warehouse when the system reads a page. However, there are two differences: 1) the interval of a continuous query will be different from the crawler interval (the user will not know anything about crawler rate for a particular page, so this interval can in fact be shorter as well as longer) and 2) the monitoring query language will often be more primitive than a general query language because of performance considerations.
2. Queries that consider contents in more than one document, but only the most recent version of the documents.
3. Queries that also consider contents in previous versions of documents. For these queries, we need a temporal query language.
4. Queries that also consider the *results* from previous queries. This can be a very powerful feature. Fortunately, it can easily be provided if the temporal features are already supported. It can be implemented by storing query results as versioned XML documents, and perform queries on these *query result documents*.

When supporting the last two query classes, we meet the problems discussed in this paper, and have to resolve these.

3.4 Vacuuming

Even though storage cost is decreasing, storing an ever growing database can still be too costly in many cases. A large database can also slow down the speed of the database system, for example because of the size of the indexes. As a consequence, it is desirable to be able to physically

delete non-current document versions and deleted documents. This process is called *vacuuming*. Note that the term *vacuuming* is also used as a term for the migration of historical data from secondary storage to cheaper tertiary storage. In this paper, we will use the term for *physical deletion* only.

4 Meeting the challenges

Based on the discussion in the previous sections, we identify some additional requirements that should be supported in an XML-DW:

- Snapshot queries, element history queries and changes.
- Time approximations.
- Inconsistent data.
- Summary data and vacuuming.

Snapshot queries, element history queries and changes can be managed by introduce explicitly snapshot time as described in [8]. How to solve the other issues will now be described in more detail.

4.1 Different time approximations

Frequently, it is possible to know what time approximation/version approximation that is reasonable to use. In this case, this information can be included in the query. This can be done by extending the time expression which as previously described had the form `[`t`]` or `[EVERY]`, where `t` is a timestamp:

- `[`t `MOST_RECENT]`: Assume the contents for all t where $t_1 < t \leq t_2$ is the same as the contents at time t_1. We expect this to be the most common time approximation, and it is also similar to the previous approach. Therefore, it should be possible to denote by the short form `[`t`]`.
- `[`t `NEAREST]`: Choose document version with timestamp t_i which minimizes $|t - t_i|$.
- `[`t`]` Use an interpolated value for the contents at t. One example is the temperature in Paris, another is the exchange rates for the dollar.
- `[`t `GRANULARITY` g`]`: Assume the contents is only valid for the times retrieved, with a given granularity g. The granularity g can for example be `DAY` (difference in time is less than 24 hour) or `WEEK` (difference in time less than 7 days). The document is assumed undefined/non-existing for all other times.

In Section 3.2.1, another solution was also mentioned: interpolating values between versions. That is an approach that could prove very useful. However, it is be more awkward to express how the interpolation should be performed, rather than simply deciding which version to use. It is possible that this task should be left to the applications, who can do this job for example by using two versions that are accessed using 1) `[`t `MOST_RECENT]` and 2) a `NEXT` operator, and a filter/change operation on these versions. Note that persistent EIDs are important for this task.

4.2 Inconsistent data

Solving the problems related to inconsistency between versions in a temporal XML-DW is only partially possible. First of all, it should be noted that this problem has two variants: 1) references to certain items that can be identified automatically, e.g., references to an identifier, and 2) reference to another document based on contents, without any identifier in the reference.
In the first case, time approximation, as described above, can be used to find a candidate version. For the second cases however, it is difficult know if a particular version is actually the one originally referenced.

4.3 Vacuuming and data reduction

We have already mentioned missing (old) versions as a reason for inconsistency. Although missing data due to vacuuming is related to the issues of incomplete data, in the case of vacuuming we have control over the deletion.

4.3.1 Link-based approach

In the case of vacuuming we have control over the deletion of old document versions from the XML-DW. As a result, it is possible to avoid removing old versions that later will be queried. Thus, the problem is: how can we know which versions might be needed later?

One way to solve this problem, is to employ general garbage collection techniques. In a non-temporal context, this problem could be solved using a link-matrix based approach, making it possible to remove a version V when there are no other documents with links to it. However, in a temporal context more information has to be available, because a link is a temporal property. The straightforward solution is to use versions as granularity in the link matrix. However, this could result in a very large matrix. Although a temporal link matrix could be useful for other purposes, for the vacuuming it is sufficient to attach to each document version a list of other versions with links to it. It should be noted that this is effectively a precomputed join. It should also be noted that although this solves the problem, it means

additional overhead when a new page with lots of links is inserted into the XML-DW.

4.3.2 Reduced granularity approach

In traditional vacuuming, it is typical to delete all historical versions with a timestamp less than t. In the link-based approach described above, all historical versions with a timestamp less than t *and* not referenced from other documents are deleted. Another approach that is very applicable in an XML-DW context is to *reduce the granularity of the stored versions*. For example, if we for one month have stored 30 different versions, we could remove 26 of these, and only keep one version for each week.

4.3.3 Summary data approach

In traditional temporal database systems it is possible to store summary versions for the vacuumed data. In this way, operations like aggregate operators can still return the same results as if the vacuumed versions were still in the database.

It is possible to create summary versions for vacuumed versions in a XML-DW as well. However, less is known about the semantics, and the fact that the data is semistructured. As a result, it is more difficult to know what kind of queries will be subsequently used, and how to actually create the summary versions. Therefore, we expect it will be necessary for the application developer to write "summary version creation functions" if summary versions are required.

5 Summary

We have in this paper studied some of the challenges we will meet in a temporal XML-DW. We have described solutions for some these challenges, but also identified some problems that are difficult to solve automatically. However, we expect that in practice, these problems will be easier to solve with the advent of more standardized DTD for specific domains, and various forms of document resource description standards. It should be noted that the issues described in this paper are also applicable for general Web warehouses based on HTML web pages.

In order to facilitate further research on the topics described in this paper, we are currently working in two directions:

1. In order to gain more experience on the functional aspects, rather than efficiency, we are following a *stratum* approach (i.e., storing all versions of all documents in the database, and use a middleware layer to convert temporal query language statements into conventional statements, executed by an underlying database system), and implementing the interfaces to a temporal XML database system on top of a commercial database system, utilizing the existing support for XML data management in the system.

2. In order to study more closely the aspects of efficiency, we also work on a prototype based on a more integrated approach. This system will to a large degree be based on existing public available software modules for XML parsing, storage, and indexing, including the use of the Berkeley DB for storage and indexing.

Acknowledgments

Part of this work was done while the author was an ERCIM fellow at the Verso group at INRIA, France, and I would like to thank Vincent Aguilera and Benjamin Nguyen for useful discussions and constructive comments about the topics presented in this paper.

References

[1] S. Abiteboul, L. Segoufin, and V. Vianu. Representing and querying XML with incomplete information. In *Proceeding of PODS 2001*, 2001.

[2] C. Dyreson. Observing transaction-time semantics with TTX-Path. In *Proceedings of the Second International Conference on Web Information Systems Engineering (WISE2001)*, 2001.

[3] C. Dyreson. Towards a temporal world-wide web: A transaction time web server. In *Proceedings of the Australian Database Conference*, 2001.

[4] F. Grandi and F. Mandreoli. The valid web: An XML/XSL infrastructure for temporal management of web documents. In *Proceedings of Advances in Information Systems, First International Conference, ADVIS 2000*, 2000.

[5] M. Liu and T. W. Ling. A data model for semistructured data with partial and inconsistent information. In *Proceeding of EDBT'2000*, 2000.

[6] B. Nguyen, S. Abiteboul, G. Cobena, and M. Preda. Monitoring XML data on the web. In *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data*, 2001.

[7] K. Nørvåg. Algorithms for temporal query operators in XML databases. In *Proceedings of Workshop on XML-Based Data Management (XMLDM) (in conjunction with EDBT'2002)*, 2002.

[8] K. Nørvåg. Temporal query operators in XML databases. In *Proceedings of the 17th ACM Symposium on Applied Computing (SAC'2002)*, 2002.

[9] L. Xyleme. A dynamic warehouse for XML data of the web. *IEEE Data Engineering Bulletin*, 24(2), 2001.