Batch Mode Update For View Maintenance Over Semi-structured Data

Dazhi Wang, Junyi Xie
{wangdz, junyi}@cs.duke.edu

Advisor: Dr. Jun Yang
junyang@cs.duke.edu

Abstract

Unlike the structured data organization in traditional relational database management system, the semi-structured data can be irregular and incomplete which is associated with schemas contained in the data itself. The materialized view for semi-structured data needs to be maintained in response to changes of base data. This report addresses the batch-mode updating problem in view maintenance over semi-structured data. We extend the base view maintenance algorithm from [1]. We propose a batch mode update algorithms that does not incur dependence problem when view is updated out of the order of base data updates. Our initial experiments show that when the number of updates is small, overhead of batch mode update as block I/O is on the same level of B+ tree index and hash index. And it takes around two orders of magnitude less access time than these two index structures when number of updates increase to more than 100,000. Additionally, it does not require special index structure if the semi-structured data is stored using relational database which saves the graph structured data by edge.

Introduction

The fact that most data source from web does not conform to traditional relational data model leads to aggressive research work on semi-structured data. Currently most research in this area is aimed at extending database management techniques to semi-structured data. One of these is the incremental view maintenance problem over semistructured data. View over base data can be used to filter the data and restructure it. And views are often materialized to speed up the response time of queries from users when underlying data is remote or response time is crucial. The key problem of view maintenance is that how to keep data in view consistent with the base data. That means when the base data updated, how to update view accordingly. View recomputation from scratch is the simplest way but the most expensive way so it is not realistic to re-compute view in reality. Many viw maintenance techniques have been explored in relational database. The main idea is that by only computing the incremental updates to the view based on the updates to the database, the view maintenance work will become much cheaper than re-computation

from scratch. However, unlike the mature incremental view maintenance algorithms for relational database, the incremental view maintenance algorithms for semi-structured data are far from perfect and optimality. There are several reasons. First, the semi-structured data is "schema-less" data, which means it does not have any fixed schema and well-defined key constraints for the stored data. Without schema, the view maintenance algorithms in relational database, which heavily depend on referential integrity do not hold any longer in scenario of semi-structured data. Second, there is no uniform data model for semi-structured data so far. Each model puts some kind of constraint over the data stored, for example, WHAX data model regulates that every edge from a node can be identified by a key value called local identifier. Therefore view maintenance algorithms over one semi-structured data model does not hold on other semi-structured data model in general. And, there is no uniform query language defined for semi-structured data. In contrast to the case of relational database in which SQL is the standard query langue, there are many different and incompatible query languages in semi-structured data, such as XML-QL, Lorel, XQL, XSLT, etc. In 1998, Abiteboul el proposes a simple view specification mechanism and an algorithm for incremental view maintenance over semi-structured data. The data model used is Objective Exchange Model(OEM). However, the algorithms proposed in that paper has some weaknesses that the author left for future work. One of these weaknesses is that it does not support batch-mode view update. As consequence, each update to base data will trigger at lease one view update query evaluation and view is updated in sequence of base update order. In this project, we are aimed at, first to propose an alternative algorithm which supports view update for a set of base data updates, second, it can reduce update cost in batch mode and we discuss the dependence problem to make sure it will not jeopardize the consistency between view and base data since the order of view update is not consistent with order of base data update. The report organizes as following, we first introduce the data model, query language in lorel system. Then we introduce how view is incrementally maintained using the base algorithm introduced in [1]. In next section we introduce the batch-mode update algorithm and discuss the update dependence problem. We then presents the simulation results we have so far to compare the performance between non-batch mode update and batch-mode update. At last, we conclude this report by some observations from this project and some future work.

Base View Maintenance Algorithms

OEM Data Model

In semi-structed data, the database can be described as a labeled, directed graph(OEM, Object Exchange Model) in which each vertex in the graphs represents an object and each object has a unique object identifier (oid). The object can be either atomic object or complex object. OEM is designed to handle the incompleteness as well as the structural and type heterogeneity of data. Figure 1 is a simple example of OEM database, in which semantic information is included in the labels as part of the data and can be exchanged dynamically. In this respect, this OEM graph is self-describing. Figure 1 describes a simple bibliographic database. It is easy to see that there is no schema in database and except the unique object identifier (oid) there is no way to globally identify the objects in graph.

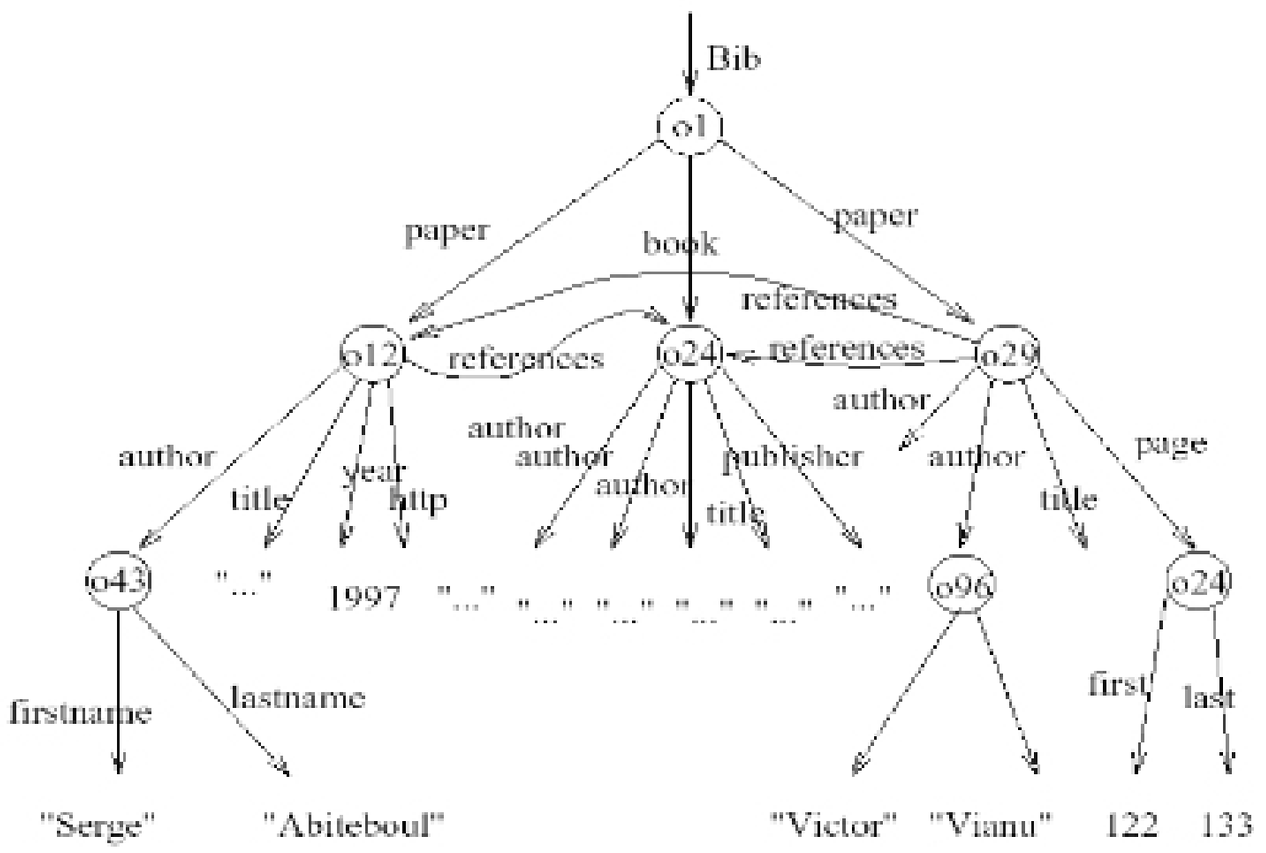

Figure 1 Sample OEM Data Model

Lorel Query Language

The Lorel query language uses the familiar *select-from-where* syntax of SQL and can be considered as extension to OQL that provides powerful path expression for traversing the data and extensive coercion rules for a more forgiving type system. Following example is a query that returns all *Entrée* sub-objects of a resteraunt named "China Hunan" and one of its ingredients has the value "Mushroom"

Exmaple 1

Select *e*
From *Guide.Restaurant r, r.Entrée e*
Where *r.Name = "China Hunan"*
and *e.Ingredient = "Mushroom"*

The path expression in lorel query language is composed of a set of one step path which has the form $x.L\ y$ where x and y are variables bound to object ids and L is variable bound to edge label.

View Definition in Lorel

The view definition in Lorel can import objects and edges from a source database into a view and new objects and edges can be created in the view. Additionally the view specification extends *select-from-where* statement with a *with* clause. Each object and edge along a path in the *with* clause is included in the view. In other words, *select-from-where* can only return a set of objects and *with* clause imports some structure into view. Example 2 defines a view as a result of example 1.