Cost-Driven General Join View Maintenance over Distributed Data Sources [*]

Bin Liu and Elke A. Rundensteiner
Department of Computer Science
Worcester Polytechnic Institute, MA 01609, USA
{binliu | rundenst}@cs.wpi.edu

Abstract

Maintaining materialized views that have join conditions between arbitrary pairs of data sources possibly with cycles is critical for many applications. In this work, we model view maintenance as the process of answering a set of inter-related distributed multi-join queries. We illustrate two strategies for maintaining as well as optimizing such general join views. We propose a cost-driven view maintenance framework which generates optimized maintenance plans tuned to a given environmental settings. This framework can significantly improve view maintenance performance especially in a distributed environment.

1 Introduction

Materialized views [4] defined over distributed data sources are widely used in many applications to ensure efficient access, reliable performance and high availability. To realize such benefits, materialized views must be maintained upon source changes since stale view extents may not serve well or even mislead user applications. Incremental view maintenance, aimed at only computing the delta changes of the view extent under source changes instead of recomputing from scratch, has been extensively studied in the literature [11, 1]. Among these works, the incremental maintenance of batches of updates [10, 3, 6, 7] is of particular interest since it is attractive to most practical systems from both a resource and a performance perspective.

State-of-the-art view maintenance algorithms [1, 10, 6, 7] usually focus on maintaining simple acyclic join views. Less attention has been paid to maintain and optimize general join views (i.e., cyclic join views that may specify multiple join conditions between any two source relations) or to exploit dynamic environmental settings such as source updates and network costs. In this work, we illustrate that such a general join view offers opportunities of more flexible maintenance strategies and thus demands the optimizations. These optimizations have the potential to improve maintenance performance significantly.

[*]This work was supported in part by the NSF grant #IIS 9988776.

2 View Maintenance and Optimization

We first illustrate the state-of-the-art view maintenance process. Assume the materialized view V is defined on 4 data sources represented as $R_1 \bowtie R_2 \bowtie R_3 \bowtie R_4$. ΔR_1, ΔR_2, ΔR_3, ΔR_4 are the corresponding source *deltas* that need to be maintained. The changes to the view extent (ΔV) by all these source deltas can be computed by Equation (1) [6]. Here R_i represents the pre-state of the underlying data source, while R'_i = R_i + ΔR_i is the post-state of the data source. We refer to each line in Equation (1) as a *maintenance step*, e.g., $\Delta R_1 \bowtie R_2 \bowtie R_3 \bowtie R_4$, and each join operation within such a step as a *maintenance query*, e.g., $\Delta R_1 \bowtie R_2$.

$$\begin{aligned} \Delta V \quad &= \Delta R_1 \bowtie R_2 \bowtie R_3 \bowtie R_4 \\ &+ R'_1 \bowtie \Delta R_2 \bowtie R_3 \bowtie R_4 \\ &+ R'_1 \bowtie R'_2 \bowtie \Delta R_3 \bowtie R_4 \\ &+ R'_1 \bowtie R'_2 \bowtie R'_3 \bowtie \Delta R_4 \end{aligned} \tag{1}$$

We view the above maintenance process (the evaluation of Equation 1) as the process of answering multiple inter-related distributed queries. Thus, the following two general optimization opportunities are available. They have not been carefully exploited in the view maintenance context.

Choose Optimized Join Ordering. Multiple ways of executing each maintenance step exist. For example, we can evaluate either $\Delta R_2 \bowtie R_3$ or $\Delta R_2 \bowtie R'_1$ first for the second maintenance step that contains ΔR_2. Different join orderings brings variations such as intermediate results that affect the overall performance. Thus, the selection of an optimal join ordering for a multi-join query, which is investigated in traditional distributed query optimization such as in [5], could be applied here to improve the maintenance performance.

Reduce Number of Maintenance Queries. Reducing the number of maintenance queries to remote data sources also has the potential to improve the maintenance performance. We propose a *grouping maintenance* algorithm [9] that maintains a materialized view defined as $R_1 \bowtie R_2 \bowtie \ldots \bowtie R_n$ using 2*(*n*-1) maintenance queries only. This reduction in the number of maintenance queries (from O(n^2) to O(n)) has been shown to be efficient. However, an important requirement for this grouping maintenance is the existence of a join condition between each R_i and R_{i+1}. Otherwise, we may be forced to evaluate a Cartesian product [9].

3 General Join View Maintenance

Considering view definitions beyond simple acyclic join views, i.e., those having multiple join conditions between arbitrary data sources possibly with cycles, new maintenance and optimization strategies can be exploited. We use *view graphs* to represent such general join view definitions. Each node represents a data source while an edge indicates a join condition between two sources. Figure 1(a) shows an example of a view graph that has 4 data sources. We describe two types of maintenance below that apply the optimizations described in Section 2 respectively.

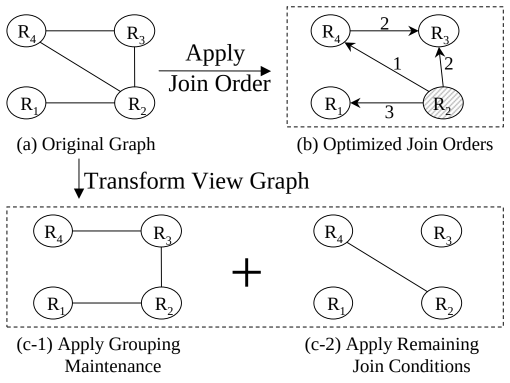
R4
R3
R1
R2
Apply
Join Order
2
1
2
3
(a) Original Graph
(b) Optimized Join Orders
Transform View Graph
+
(c-1) Apply Grouping Maintenance
(c-2) Apply Remaining Join Conditions

Figure 1. Maintaining General Join Views

General join view maintenance applying join ordering optimization reduces to seeking optimized join orders for each maintenance step. For example, Figure 1(b) illustrates one possible processing of the maintenance step containing ΔR_2 for the view graph defined by Figure 1(a). The number listed on each edge indicates the ordering. Note that multiple join conditions can be combined and evaluated at the same time. As seen in Figure 1(b), once we evaluate $\Delta R_2 \bowtie R_4$, then we can incorporate the join conditions indicated by edges $R_2 - R_3$ and $R_4 - R_3$ and evaluate both of them (against R_3) at the same time.

General join view maintenance with grouping optimization reduces to transforming the join graph into simple subgraphs and applying the grouping maintenance whenever possible. As an example, the view graph of Figure 1(a) can be divided into two simpler subgraphs as shown in Figures 1(c-1) and (c-2). We can apply grouping maintenance to the part indicated by $R_1 - R_2 - R_3 - R_4$ (Figure 1(c-1)). The remaining join condition(s) can be applied after we get the result of $R_1 - R_2 - R_3 - R_4$. Due to space limitations, we ask readers to refer [8] for details.

4 Cost-based Optimization Framework

We enhance the view graph by incorporating relevant cost information for estimating the cost of maintenance process applying above maintenance and optimization strategies. Algorithms have been proposed to generate optimized view maintenance plans that incorporate the impact of view definitions as well as environmental settings such as network cost and data source processing capabilities [8].

A view maintenance optimization framework has been implemented based on the TxnWrap [2] system. Experimental results show that the view maintenance performance can be improved significantly by using our proposed optimization strategies. While we omit details due to space limitations, readers may refer to our technical report [8] for more information.

To conclude, this work illustrates effective view maintenance optimization strategies for general join view definitions. Our work also brings two independent areas one step closer, namely, the fusion of distributed query optimization and view maintenance optimization.

References

[1] D. Agrawal, A. E. Abbadi, A. Singh, and T. Yurek. Efficient View Maintenance at Data Warehouses. In *SIGMOD*, pages 417–427, 1997.

[2] S. Chen, B. Liu, and E. A. Rundensteiner. Multiversion Based View Maintenance over Distributed Data Sources. *ACM TODS*, 2004, to appear.

[3] L. S. Colby, T. Griffin, L. Libkin, I. S. Mumick, and H. Trickey. Algorithms for Deferred View Maintenance. In *SIGMOD*, pages 469–480, 1996.

[4] A. Gupta and I. Mumick. Maintenance of Materialized Views: Problems, Techniques, and Applications. *IEEE Data Engineering Bulletin*, 18(2):3–19, 1995.

[5] D. Kossmann. The State of the Art in Distributed Query Processing. *ACM Computing Surveys (CSUR)*, 32(4):422–469, 2000.

[6] W. J. Labio and R. Y. and. Shrinking the Warehouse Updated Window. pages 383–395, June 1999.

[7] B. Liu, S. Chen, and E. A. Rundensteiner. Batch Data Warehouse Maintenance in Dynamic Environments. In *CIKM'02*, pages 68–75, Nov 2002.

[8] B. Liu and E. A. Rundensteiner. Cost-Driven View Maintenances in Distributed Environments. Technical Report WPI-CS-TR-03-30, WPI, 2003.

[9] B. Liu, E. A. Rundensteiner, and D. Finkel. Restructuring View Maintenance Plans for Large Update Batches. In *CIKM, Poster*, page to appear, 2004.

[10] K. Salem, K. S. Beyer, and et.al. How To Roll a Join: Asynchronous Incremental View Maintenance. In *SIGMOD*, pages 129–140, 2000.

[11] Y. Zhuge, , J. Hammer, and J. Widom. View Maintenance in a Warehousing Environment. In *SIGMOD*, pages 316–327, May 1995.