Restructuring Batch View Maintenance Efficiently *

Bin Liu, Elke A. Rundensteiner and David Finkel
Worcester Polytechnic Institute, MA, 01609 USA
{binliu | rundenst | dfinkel}@cs.wpi.edu

ABSTRACT

Materialized views defined over distributed data sources are a well recognized technology for modern applications. State-of-the-art incremental view maintenance requires $O(n^2)$ or more maintenance queries with n being the number of data sources in the view definition. In this work, we propose novel maintenance strategies that dramatically reduce the number of maintenance queries to remote data sources. The proposed algorithms have been implemented in a working prototype system. Experimental studies illustrate that our algorithms is able to achieve about 400% performance improvement in terms of total processing time compared with existing batch algorithms.

Categories and Subject Descriptors: H.2.4 [Database Management]: Systems–Distributed databases

General Terms: Algorithms, Performance.

Keywords: Batch Maintenance, Grouping Maintenance.

1. INTRODUCTION

Materialized views [1, 4] that integrate and store data from distributed data sources are applied to ensure better access, reliable performance and high availability. They have been widely used in applications such as decision support system, data warehousing and e-business. Materialized views must be maintained upon source changes in order to provide quality results. Incremental view maintenance aims at only computing the deltas of the view result instead of recomputing the view from scratch [2, 9, 1, 8, 3]. Among these works, the incremental maintaining of batches of updates [8, 6, 5] is of particular interest because it is attractive from both a resource and a performance perspective to most practical systems. are two fold. One,

However, modern data sources are becoming increasingly large. Rapid changes made to such data sources are common too. Moreover, the data sources tend to be distributed over the network. These trends of most practical systems pose new challenges to efficient materialized view maintenance. State-of-the-art view maintenance strategies require $O(n^2)$ (for batch view maintenance) or more (i.e., for sequential maintenance) *maintenance queries* [9] to remote data sources with n being the number of data sources in the view definition. They usually only batch the updates from the same data source [8, 6, 5]. This mechanism does not scale for large sized nor for large number of data sources. In this work, we investigate scalable view maintenance strategies. The basic idea is to restructure the batch view maintenance aimed at reducing the number of maintenance queries to remote data sources. Though such reduction in the number of maintenance queries will increase the complexity of each such query, we find that it still outperforms existing batch view maintenance strategies in a significant manner (around 400% improvement) in a majority of the cases.

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

2. STATE-OF-THE-ART

We use the following example to describe the state-of-the-art incremental view maintenance. 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 delta changes that need to be maintained. The changes to the view extent (ΔV) by all these source delta changes can be computed by Equation (1) [5]. Here R_i represents the pre-state of the underlying data source, while $R_i' = R_i + \Delta R_i$ 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., $\Delta R_1 \bowtie R_2$.

$$\begin{aligned} \Delta V &= \Delta R_1 \bowtie R_2 \bowtie R_3 \bowtie R_4 \\ &+ R_1' \bowtie \Delta R_2 \bowtie R_3 \bowtie R_4 \\ &+ R_1' \bowtie R_2' \bowtie \Delta R_3 \bowtie R_4 \\ &+ R_1' \bowtie R_2' \bowtie R_3' \bowtie \Delta R_4 \end{aligned} \quad (1)$$

Equation 1 requires $O(n^2)$ remote maintenance queries to compute ΔV where n is the number of data sources.

3. GROUPING MAINTENANCE

Adjacent Grouping. One method to reduce the number of maintenance queries is to share the common access to the data sources. As illustrated in Figure 1, we could divide the four maintenance steps into two groups. The first two maintenance steps then can be rewritten into $(\Delta R_1 \bowtie R_2 + R_1' \bowtie \Delta R_2) \bowtie R_3 \bowtie R_4$, while the other two can be rewritten into $(\Delta R_3 \bowtie R_4 + \Delta R_4 \bowtie R_3') \bowtie R_2' \bowtie R_1'$. Thus, the total number of maintenance queries will be reduced from 12 to 8 in this case. In general, if we divide maintenance

steps equally, the total number of maintenance queries will be $O(n^{3/2})$ [7].

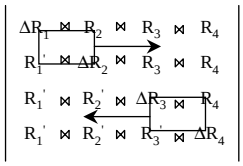

Figure 1: Adjacent Grouping

Conditional Grouping. A more aggressive grouping scheme can compute ΔV using just $2*(n-1)$ maintenance queries when the view is defined as a linear join ($V = R_1 \bowtie R_2 \bowtie \ldots \bowtie R_n$), namely, a join condition(s) between each adjacent pair R_i and R_{i+1}.

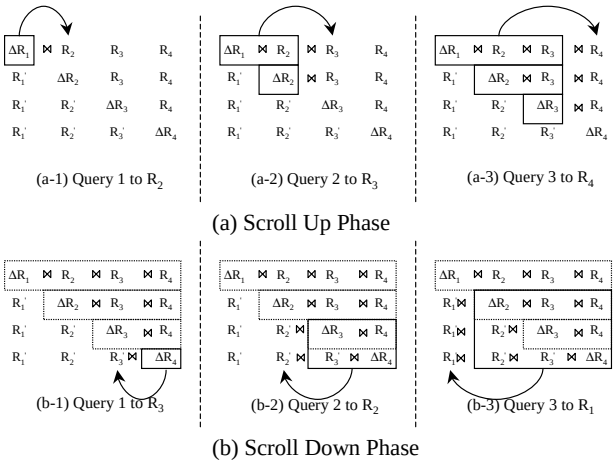

Figure 2: Conditional Grouping

We now describe this grouping strategy based on the above example view definition. There are two phases for this grouping strategy. The first is the *scroll up* phase (Figure 2(a)), which sends ΔR_1 to data source R_2 and evaluates the result $\Delta R_1 \bowtie R_2$. After we get the result, we union the result with ΔR_2 and generate a combined maintenance query to data source R_3. We get $\Delta R_1 \bowtie R_2 \bowtie R_3$ and $\Delta R_2 \bowtie R_3$ as the second query result. Then we union ΔR_3 into this second query result and compose the maintenance query to R_4. The result of the scroll-up phase is $\{\Delta R_1 \bowtie R_2 \bowtie R_3 \bowtie R_4, \Delta R_2 \bowtie R_3 \bowtie R_4, \Delta R_3 \bowtie R_4\}$. The *scroll down* phase (Figure 2(b)) computes the remaining part of the queries. As in Figure 2(b), the first maintenance query in this phase evaluates $\Delta R_4 \bowtie R_3'$. The second maintenance query combines the result of the first query ($\Delta R_4 \bowtie R_3'$) with the corresponding part of the scroll-up phase result ($\Delta R_3 \bowtie R_4$) and evaluates them against $\Delta R_2'$. We go on with this process until we reach the data source R_1 to get the final result of ΔV. Thus, the total number of maintenance queries in this case is reduced from 12 to 6.

Details of the algorithm such as how to group different deltas and how to maintain views beyond simple linear joins are omitted here due to space limitation. Readers can refer [7] for in-depth discussions.

4. EXPERIMENTS

We have implemented above maintenance strategies within the TxnWrap system [3]. The experiments are conducted on four Pentium III 500MHz PCs connected via a local network with 512M memory, running Windows 2000 and Oracle 8*i*. We employ six data sources with one relation each over three PCs. Each relation has 1M tuples with 64 bytes on average of each tuple size. A materialized join view is defined through equi-joins upon these six source relations residing on the fourth machine. The view contains 1M tuples with each tuple having 384 bytes on average. All the source deltas are composed of approximately the same number of insert and delete tuples. Two queries are needed when a single delta contains both insert and delete tuples.

	1k	5k	10k	15k	20k	25k
Batch (s)	821.7	842.7	876.7	947.8	1031.2	1108.4
Cond. (s)	247.6	260.2	289.4	316.3	357.3	437.1

Due to space constraints, we only compare the performance of batch and conditional grouping given the total number of source updates changing from $1k$ to $25k$. A comprehensive study can be found in [7]. Seen from the above table, we can see that the conditional grouping outperforms batch maintenance due to the conditional grouping having a smaller number of maintenance queries.

5. CONCLUSION

We have proposed two novel maintenance algorithms that require a smaller number of maintenance queries to remote data sources. The experimental studies illustrate that maintenance performance can be significantly improved by our proposed solutions compared with the typical batch maintenance in the literature.

6. REFERENCES

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

[2] J. A. Blakeley, P.-A. Larson, and F. W. Tompa. Efficiently Updating Materialized Views. In *Proceedings of SIGMOD*, pages 61–71, May 1986.

[3] S. Chen, B. Liu, and E. A. Rundensteiner. Multiversion Based View Maintenance over Distributed Data Sources. *ACM Transactions on Database Systems (TODS)*, 2004, to appear.

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

[5] W. J. Labio, R. Yerneni, and H. García-Molina. Shrinking the Warehouse Updated Window. In *Proceedings of SIGMOD*, pages 383–395, June 1999.

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

[7] B. Liu, E. A. Rundensteiner, and D. Finkel. Restructuring View Maintenance Plans for Large Update Batches. Technical Report WPI-CS-TR-03-29, WPI, 2003.

[8] K. Salem, K. S. Beyer, R. Cochrane, and B. G. Lindsay. How To Roll a Join: Asynchronous Incremental View Maintenance. In *SIGMOD*, pages 129–140, 2000.

[9] Y. Zhuge, H. García-Molina, J. Hammer, and J. Widom. View Maintenance in a Warehousing Environment. In *Proceedings of SIGMOD*, pages 316–327, May 1995.