SWECCA for Data Warehouse Maintenance*

Aparna Varde and Elke Rundensteiner
Department of Computer Science, Worcester Polytechnic Institute
Worcester, MA 01609, USA

Abstract

A Data Warehouse is a customized repository of integrated information catered to the needs of the user. Warehouses today typically draw information from several sources each with numerous relations. Efficient maintenance of such warehouses with up-to-date information poses challenges. We face the problem of concurrency conflicts in information sources during incremental View Maintenance (VM). Some existing VM algorithms partly solve this problem, either in a single-source warehouse or in a multi-source warehouse with only one relation per source. We propose the SWECCA approach that combines techniques from a multi-source VM algorithm SWEEP and a single-source VM algorithm CCA and further refines them to provide a complete solution to the problem. SWECCA overcomes concurrency conflicts in the incremental View Maintenance of data warehouses with multi-relation information sources. It does not require the storage of any intermediate views, thus providing a space-efficient solution.

Keywords: Data Warehouse, Incremental View Maintenance, Multi-relation Sources, Distributed Databases, Concurrency Conflict, Space-efficiency.

1 Introduction

A data warehouse is a materialized repository of information collected from various sources and integrated based on the requirements of the user [CD97]. It provides at-a-glance data for applications like decision support [CD], information systems [CYC+96] and others. The warehouse has to be maintained up-to-date with changes in the sources. Owing to the large size of warehouses we prefer to maintain them *incrementally* [MKK97]. Incremental View Maintenance (VM) algorithms like [Zhu99, AESY97] have to deal with data updates occurring concurrently and affecting each other.

*This work was supported in part by the NSF NYI grant #IRI 97–96264, the NSF CISE Instrumentation grant #IRIS 97–29878, and the NSF grant #IIS 9988776.

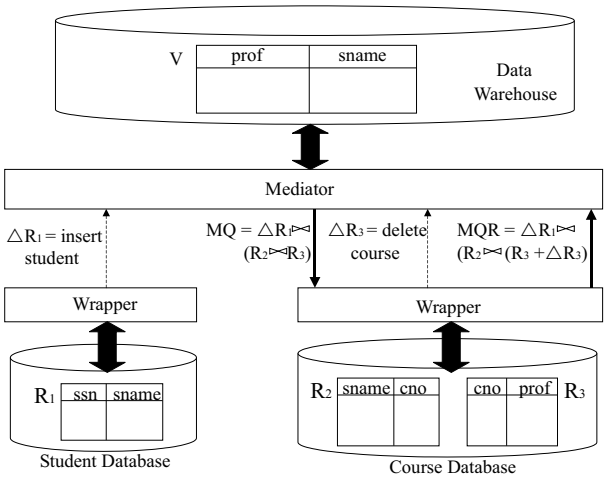

Figure 1: Illustration of a Concurrency Conflict in Data Warehouse Maintenance

For example, in the data warehousing system in Figure 1,[1] if update ΔR_1 occurs, say a new student is added to the Student database, then a *maintenance query* (MQ) [AESY97] is sent to the Course database to find out what joins with this student in the Course relations R_2 and R_3. But the maintenance query result MQR includes the effect of update ΔR_3, say a course getting dropped, since ΔR_3 occurs during the processing of the MQ. This is wrong, because the dropped course should include the effect of the new student, and not vice versa since the new student got added before the course got dropped. The discrepancy is due to the course delete being concurrent with the maintenance process of the student insert. This problem is called a *concurrency conflict.*

1.1 Existing VM Algorithms

CCA [Zhu99] solves this problem, but only in a single-source data warehouse with all relations in the same source. SWEEP [AESY97] deals with concurrency conflicts allowing the relations to be spread over multiple sources, however assuming only one re-

[1] In all figures, except Figure 4, arrows from left to right indicate the order of message-passing as in a timing diagram.

lation per source. These and other VM algorithms like [ZGMHW95, ZGMW96] have their shortcomings. More on this in Section 6. Warehouses today typically draw their information from multiple sources, each with multiple relations. We need a solution for concurrency conflicts in such environments.

1.2 Our Proposal

Since CCA [Zhu99] does VM in a single-source environment, and SWEEP [AESY97] in a multi-source environment with one relation per source, we propose to combine ideas from both and add further refinement to solve concurrency conflicts in the incremental maintenance of warehouses with multi-relation sources. We thus propose the SWECCA approach that uses SWEEP-techniques at the mediator[2] of the system, and CCA-techniques at the wrapper[3] of each source. SWECCA maintains the warehouse perfectly up-to-date with changes in the sources, i.e. it has *complete consistency* as defined in [ZGMW96]. It does not require the storage of any intermediate views, thus providing space-efficiency.

1.3 Paper Layout

Section 2 provides background material on the SWEEP and CCA algorithms. SWECCA is proposed in Section 3 with its architecture in Section 4. Section 5 gives an application standpoint. Section 6 summarizes related work. Section 7 gives conclusions.

2 Background Material

2.1 The CCA Algorithm

CCA [Zhu99] was proposed at Stanford University in 1995 for view maintenance in a single-source data warehouse. It assumes that all relations are in the same information source. It uses a remote compensation technique to correct concurrency conflicts, and does VM with complete consistency [ZGMW96].

In the CCA system shown in Figure 2, when a data update say ΔR_1 occurs, the mediator sends a maintenance query $MQ = \Delta R_1 \bowtie R_2 \bowtie R_3$. If updates ΔR_2 and ΔR_3 occur between the MQ and its result MQR, then they cause concurrency conflicts. Thus incorrect $MQR = \Delta R_1 \bowtie (R_2 + \Delta R_2) \bowtie (R_3 + \Delta R_3)$ is returned. The mediator sends a remote compensating query in response to each conflict to find out what correction needs to be applied due to the conflict. Thus

[2]The Mediator is the data warehouse manager that maintains the warehouse once it is created, in keeping with changes in the sources.

[3]Each source in a data warehousing system has a Wrapper that extracts data from the source, cleans it and converts it to a form suitable for the warehouse.

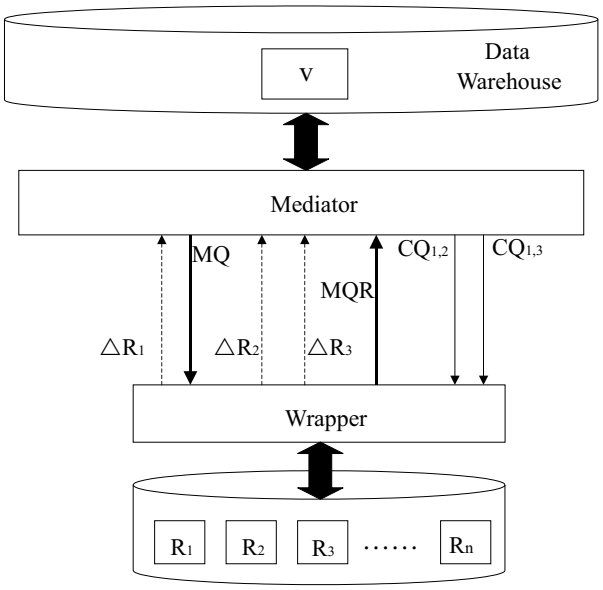

Figure 2: Single-source Data Warehouse with the VM Algorithm CCA

$CQ_{1,2} = -(\Delta R_1 \bowtie \Delta R_2 \bowtie R_3)$. Since ΔR_3 occurs after ΔR_2, the compensating query for the latter takes into account the effect of the former. Thus $CQ_{1,3} = -(\Delta R_1 \bowtie R_2 \bowtie \Delta R_3) + (\Delta R_1 \bowtie \Delta R_2 \bowtie \Delta R_3)$. The warehouse view is updated using all the query answers, i.e. $\Delta V = MQR + CQR_{1,2} + CQR_{1,3} = \Delta R_1 \bowtie R_2 \bowtie R_3$ which is correct.

CCA works using an event-based approach. A Source Update event S-Up sends data updates from the source to the mediator or warehouse manager. A Warehouse Update event W-Up sends maintenance queries to the source in response to updates. A Source Query event S-Qu answers maintenance queries, and sends results to the warehouse manager. A Warehouse Answer event W-Ans sends compensating queries if there are concurrency conflicts and uses all query answers to maintain the view at the warehouse. For details, refer to [Zhu99].

The CCA algorithm however is not designed for VM across a multi-source warehousing environment.

2.2 The SWEEP Algorithm

SWEEP [AESY97] was proposed at the University of California in Santa Barbara in 1997. It assumes that the relations are spread across multiple sources with only one relation per source. It works using a local compensation approach, and maintains the warehouse with complete consistency [ZGMW96].

In the SWEEP system shown in Figure 3, when a data update say ΔR_1 occurs, SWEEP sends a maintenance query (MQ) independently to each information source except the one from where the data update ar-

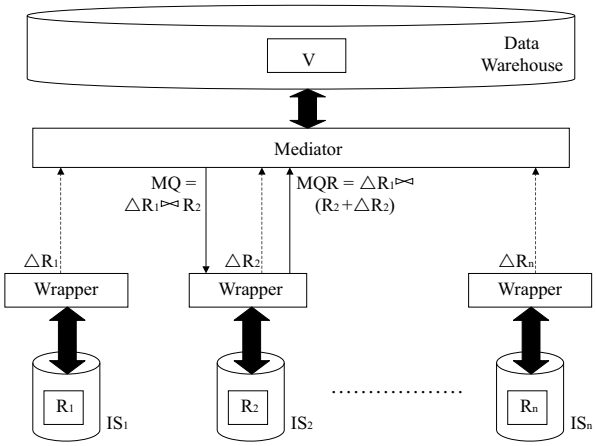

Figure 3: Multi-source Data Warehouse with the VM Algorithm SWEEP

rived. The MQ finds out what joins with the update in that respective source. Here $MQ = \Delta R_1 \bowtie R_2$ for source IS_2. It gets the maintenance query result (MQR) in its update message queue (UMQ). If there are other updates in the queue between an MQ and its MQR, then these cause concurrency conflicts, giving an incorrect MQR. Thus here $MQR = \Delta R_1 \bowtie (R_2 + \Delta R_2)$. Local compensation is used at the mediator to overcome the effect of these conflicts by subtracting the appropriate values of the update notifications in the UMQ from the incorrect MQR. Details are found in [AESY97].

It has been observed that the local compensation feature of SWEEP fails if there are multiple relations per source, unless the mediator treats every relation as a separate source, posing tremendous overhead.

3 The SWECCA Approach

SWEEP [AESY97] and CCA [Zhu99] are both VM algorithms that deal with concurrent data updates using compensation. We discussed their drawbacks in Sections 2.1 and 2.2 respectively. We need an efficient solution for concurrency conflicts in a multi-source warehouse with multiple relations per source. The SWECCA approach provides this solution.

3.1 The Main Principle

From Section 2.2 it is clear that in a multi-source environment, if mediator conceptually sees each source as one relation, then it can use SWEEP [AESY97] as a VM algorithm. To achieve this we need to propagate an effective *source update* or ΔIS corresponding to every *relational update* or ΔR. If this ΔIS is built after compensating for all the concurrency conflicts within that source, then the mediator can carry out further VM from there. However we do not want wrappers to store materialized views or cartesian products of all relations in the source, since this would be analogous to having a mini-warehouse at each wrapper causing huge space overhead. Thus we need a technique that builds a ΔIS from a ΔR without intermediate storage.

The CCA algorithm [Zhu99] is useful here. The VM techniques used by CCA in a single-source environment to build a view refresh or ΔV (corresponding to a relational update ΔR) can be used to build a source update ΔIS instead. CCA can thus be used at the wrapper of each source to make it conceptually appear as one relation without intermediate storage. The CCA warehouse events W-Up and W-Ans thus become wrapper events Wrap-Up and Wrap-Ans, and the source events S-Up and S-Qu remain almost the same. With CCA at each wrapper, SWEEP can now function as before at the mediator.

This is the main principle behind SWECCA. However on a deeper level there are issues related to query processing and concurrency conflicts.

3.2 Global Query Processing

A Global Query or GQ in SWECCA is analogous to a maintenance query in SWEEP [AESY97]. It is a query generated by the mediator in response to a source update ΔIS to find out what joins with this update in each of the other sources, based on the view definition V at the warehouse. Similarly, a Local Query or LQ in SWECCA is a query generated by the wrapper, using CCA [Zhu99], in response to a relational update ΔR in the source, to find out what joins with this update in the other relations of that source, based on the projection of the view definition V of the mediator at this source. This is essential to generate a ΔIS from a ΔR.

However the original CCA algorithm, though capable of generating its own local queries, is not designed to answer global queries coming from an external entity, in this case the mediator. Hence we need additional functionality in each wrapper to process global queries. A simple method would be the wrapper accepting global queries from the mediator, directing them to the source, receiving the query answers and forwarding them to the mediator. But while doing this, data updates may occur in that source between the query and its result, leading to concurrency conflicts and affecting the query answer. This is similar to the concurrency conflict problem during view maintenance. Therefore this simple method is not feasible.

The compensation techniques used for update processing to build a ΔIS from a ΔR need be applied to global query processing as well. Hence the wrapper needs a common queue for all types of messages since

the same concurrent data update could cause a conflict in both update processing and query processing.

4 SWECCA Architecture

Figures 4 and 5 show SWECCA at the wrapper and mediator respectively.

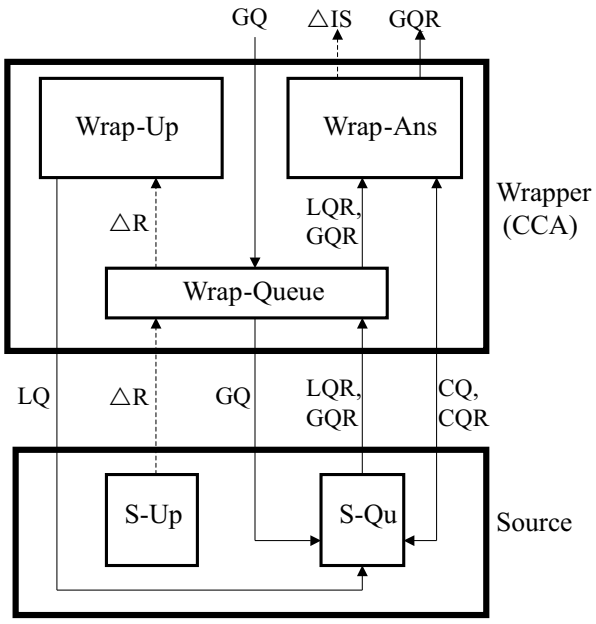

Figure 4: The SWECCA Wrapper

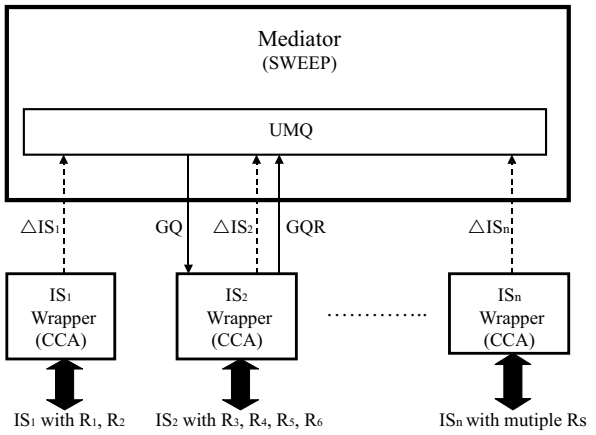

Figure 5: The SWECCA Mediator

Wrapper: Referring to Figure 4, the *Wrap-Queue* is a common queue to store all messages in the wrapper. It accepts relational updates from the source and global queries from the mediator and forwards them to the appropriate components for processing. It also stores answers to local and global queries as they arrive. The *S-Up* event sends a relational update to the wrapper and triggers the *Wrap-Up* event. This generates a local query in response to the relational update and triggers *S-Qu*. The *S-Qu* event can now get triggered by a local or a global query. It generates the corresponding query results and triggers the *Wrap-Ans* event, that detects concurrency conflicts if any. A concurrency conflict is identified as a relational update that enters *Wrap-Queue* after a local or global query is sent and before its result arrives. *Wrap-Ans* sends a compensating query for each conflict. It gets all query answers and sends the final output, either a source update ΔIS or a global query result GQR, to the mediator.

Mediator: From Figure 5, the mediator on receiving a source update (e.g., ΔIS_1) sends a global query to each source (e.g., to IS_2). The global query result arrives in the update message queue UMQ. If meanwhile, another source update (e.g., ΔIS_2) arrives in UMQ after the global query and before its result, then this is identified as a concurrency conflict at the mediator level, since it could cause an incorrect view refresh ΔV to be computed using SWEEP techniques. The order of messages in UMQ serves to detect concurrency conflicts. The mediator uses the local compensation feature of SWEEP to correct this. Note that this now works, since each source is now effectively treated as one relation.

SWECCA thus solves concurrency conflicts in the incremental maintenance of data warehouses with multi-relation sources, without additional storage of materialized views or cartesian products.

5 An Application Perspective

Many applications of data warehouses depend on the time-critical nature of data. Decision Support Systems (DSS) e.g. [Gro01] make decisions for the user based on current data and knowledge of the application. For example, in corporate ERP/MRP (Enterprise Resource Planning/Material Resource Planning) [Sti02], if the cost of a material goes up during decision-making, the DSS tool should be able to alter its decision accordingly. It could suggest buying a lower quantity of the material or not buying it at all or buying the same quantity as before, depending on cost-benefit analysis with the latest data.

Information Systems use knowledge from existing data sets and automate a task. Examples are Management Information Systems (MIS) [Pub02] and Global Information Systems (GIS) [LSK95]. In an MIS like an airline reservation system, a database may indicate that flights from Boston to Providence are now available. While its effect is being computed by the MIS, another database may offer discounted rates on all flights leaving Boston. Obviously, the second update must incorporate the effect of the first.

Expert Systems, e.g. [Bha02] work with triggers and active data warehouses [ST00]. Triggers [WCC] are event-condition-action rules that specify what action a database needs to take when a certain condition is satisfied on the occurrence of an event. For example, the system may activate a sensor if temperature is above a threshold to set a safety alarm. Applications such as robotics [DJ00] also involve very crucial tasks based on the most recent value of data.

Clearly, concurrency conflicts pose an issue in all the above applications.

6 Related Work

ECA [ZGMHW95] works similar to CCA [Zhu99], but does remote compensation before the maintenance query answer arrives, providing faster output at the expense of lower consistency. ECA has the restriction that all relations be in a single information source. Strobe [ZGMW96] does remote compensation in a multi-source environment, provided there is only one relation per source. Strobe and ECA both maintain only strong consistency as defined in [ZGMW96]. The materialized Multi Relation Encapsulation Wrapper (MRE) [DZR99] encapsulates all the relations in multi-relation sources into a single relation allowing existing VM algorithms to function without modification, but the storage cost at wrappers here is huge. In the Self Maintenance strategy (SM) [GJM96] the data warehouse maintains up-to-date copies of its relations using them to compute the effect of concurrent updates, causing overhead at the warehouse. In the version-control technique [CCR], the latest version of a transaction/tuple based on time stamps, is used to answer queries for the warehouse, requiring additional storage of versioned IS data and schema.

7 Conclusions

SWECCA solves concurrency conflicts in incremental view maintenance in a distributed multi-source data warehouse with multiple relations per source. SWECCA has the advantage of allowing the sources to be semi-autonomous since they are not involved in maintenance beyond reporting updates and processing queries. It offers the benefit of software reuse because techniques from existing VM algorithms are combined. Above all, it provides space efficiency since no intermediate views need to be stored. SWECCA is being enhanced in WPI to a more generic system called MEDWRAP [VR02] that explores the issues involved in reusing any single-source VM algorithm at the wrapper and any multi-source VM algorithm at the mediator, to achieve consistent view maintenance over distributed multi-relation sources.

Acknowledgements:

Our sincere gratitude to the DSRG group and in particular the Data Warehousing team at WPI namely Songting Chen, Jehad Al-Sabbah and Bin Liu for their co-operation. We would also like to thank Prof. Carolina Ruiz from WPI for her feedback during this research.

References

[AESY97] D. Agrawal, A. El Abbadi, A. Singh, T. Yurek. Efficient View Maintenance at Data Warehouses. In SIGMOD, pg.417–427, 1997.

[Bha02] S. Bhattacharya. TEST: The Expert System for Thermodynamics. In kahuna.sdsu.edu/testcenter/, 2002.

[CCR] J. Chen, S. Chen, E. Rundensteiner. TxnWrap: A Transactional Approach to Data Warehouse Maintenance. Technical Report, WPI, 2001.

[CD] S. Chaudhari, U.Dayal. Data Warehousing and OLAP for Decision Support. In SIGMOD, pg.507–508, 1997.

[CD97] S. Chaudhuri, U. Dayal. An Overview of Data Warehousing and OLAP Technology. SIGMOD Record, 26(1):pg.65–74, 1997.

[CYC+96] W. Chu, H. Yang, K. Chiang, M. Minock, G. Chow, C. Larson CoBase: A Scalable and Extensible Cooperative Information System. JIIS, 6(2/3):pg.223–259, 1996.

[DJ00] G. Dudek , M. Jenkin. Computational Principles of Mobile Robotics. Cambridge University Press, 2000.

[DZR99] L. Ding, X. Zhang, E. Rundensteiner. The MRE Wrapper Approach. In DOLAP, pg.30–35, 1999.

[GJM96] A. Gupta, H. Jagadish, I. Mumick. Data Integration using Self-Maintainable Views. In EDBT, pg.140–144, 1996.

[Gro01] The Oak Group. MCAP: Medical Decision Support System. In www.oakgroup.com/index2.html, 2001.

[LSK95] A. Levy, D. Srivastava, T. Kirk. Data Model and Query Evaluation in Global Information Systems. In JIIS, 5(2):pg.121–143, 1995.

[MKK97] M. Mohania, S. Konomi, Y. Kambayashi. Incremental Maintenance of Materialized Views. In DEXA, pg.551–560, 1997.

[Pub02] RMIS Web Group. The Internet Resource for Risk Management Information Systems. In rmisweb.com/, 2002.

[ST00] M. Schrefl, T. Thalhammer. On Making Data Warehouses Active. In DAWAK, pg.34–46, 2000.

[Sti02] K. Stille. List of Software Vendors Offering ERP/MRP/CRM Solutions. In business-soft.about.com/cs/mrperpcrm/, 2002.

[VR02] A. Varde, E. Rundensteiner. The MED-WRAP Approach. Technical Report, WPI, 2002.

[WCC] J.Widom, S. Ceri, U. Dayal. Active Database Systems: Triggers and Rules for Advanced Database Processing. Morgan Kaufmann Publishers, 1995.

[ZGMHW95] Y. Zhuge, H. García-Molina, J. Hammer, J. Widom. View Maintenance in a Warehousing Environment. In SIGMOD, pg.316–327, 1995.

[ZGMW96] Y. Zhuge, Héctor García-Molina, and J. L. Wiener. The Strobe Algorithms for Multi-Source Warehouse Consistency. In International Conference on Parallel and Distributed Information Systems, pg.146–157, 1996.

[Zhu99] Y. Zhuge. Incremental Maintenance of Consistent Data Warehouses. PhD Thesis, Stanford University, 1999.