The MRE Wrapper Approach: Enabling Incremental View Maintenance of Data Warehouses Defined On Multi-Relation Information Sources*

Lingli Ding, Xin Zhang and Elke A. Rundensteiner
Department of Computer Science
Worcester Polytechnic Institute
Worcester, MA 01609-2280
{lingli | xinz | rundenst}@cs.wpi.edu

Abstract

Some of the most recently proposed algorithms for the incremental maintenance of materialized data warehouses (DW), such as SWEEP and PSWEEP, offer several significant advantages over previous solutions, such as high-performance, no potential for infinite waits and reduced remote queries and thus reduced network and information source (IS) loads. However, similar to many other algorithms, they still have the restricting assumption that each IS can be composed of just one single relation. This is unrealistic in practice. In this paper, we hence propose a solution to overcome this restriction. The Multi-Relation Encapsulation (MRE) Wrapper supports multiple relations in ISs in a manner transparent to the rest of the environment. The MRE Wrapper treats one IS composed of multiple relations as if it were a single relation from the DW point of view; thus any existing incremental view maintenance algorithms can now be applied even to such complex ISs without any changes. Hence, our method maintains all advantages offered by existing algorithms in particular SWEEP and PSWEEP, while also achieving the additional desired features of being non-intrusive, efficient, flexible and well-behaved.

1 Introduction

Data warehousing (DW) is a popular technology to integrate data from heterogeneous information sources (ISs) in order to provide data to for example decision support or data mining applications [2]. Once a DW is established, the problem of maintaining it consistent and up-to-date with the underlying ISs remains a critical issue. It is popular to maintain the DW incrementally [1, 6, 4, 9] instead of recomputing the whole extent of the DW after IS updates due to the large size of DWs and the enormous overhead associated with the DW loading process.

*This work was supported in part by several grants from NSF, namely, the NSF NYI grant #IRI 97-96264, the NSF CISE Instrumentation grant #IRIS 97-29878, and the NSF grant #IIS 97-32897. Dr. Rundensteiner would like to thank our industrial sponsors, in particular, IBM for the IBM partnership award, and GTE for partial support of Xin Zhang.

In recent years, there have been a number of algorithms proposed for incrementally view maintenance [10, 11, 1, 8]. ECA [10] handles view maintenance under concurrent data updates of one centralized IS, while Strobe [11], SWEEP [1] and PSWEEP [8] handle distributed ISs. SWEEP and PSWEEP represent a significant improvement over Strobe in terms of performance (the number of messages and data sizes transferred between the DW and the ISs). PSWEEP is a parallelized extension of SWEEP and offers several orders of magnitude in a performance increase over SWEEP.

Furthermore, Strobe is subject to the potential threat of infinite waiting, i.e., the DW extent may never get updated. SWEEP and PSWEEP eliminate this limitation by applying local compensation techniques. This avoids the need for a quiescent state of the environment before being able to update the DW. But like many other DW maintenance algorithms, both SWEEP and PSWEEP assume that each IS only contains one single relation. This is unrealistic in practice as real data sources may contain 10, 100 or more relations. It is thus important to be able to support multiple relations for each IS.

1.1 Motivation

SWEEP and PSWEEP assume that there is only one relation per IS. Below we now demonstrate that the local correction technology they use does not work in the case of multiple relations per IS.

Example 1 *Given two information sources IS_1 and IS_2. R_1 and R_2 are relations of IS_1 and R_3 is relation of IS_2. Assume a DW defined by $V = R_1 \bowtie R_2 \bowtie R_3$. Now, there is a data update ΔR_3 of R_3 in IS_2. To calculate the incremental change of this update on the DW extent, denoted by ΔV, the SWEEP mediator would send down the following maintenance query to IS_1: $MQ = \Delta R_3 \bowtie (R_1 \bowtie R_2)$. If a data update ΔR_2 occurs while the MQ query is sent down, then the wrapper would send back the following (incorrect) query result to the mediator incorporating the concurrent data update ΔR_2: $MQR = \Delta R_3 \bowtie (R_1 \bowtie (R2 + \Delta R_2))$. The SWEEP mediator handles this problem using a local compensation (LC) strategy. This key feature of SWEEP guarantees that SWEEP can successfully avoid an infinite wait, because local queries do not raise the possibility of further concurrent data updates (DUs) requiring compensation. But $\Delta R_3 \bowtie (R_1 \bowtie \Delta R_2)$ cannot be locally compensated because we need information about the extent of R_1 for this purpose. However, R_1 is not locally available in DW, and hence to get the correct result, the mediator needs to generate a re-*

mote query and send it down to IS_1. In short, LC, the main feature of SWEEP, is broken.

One intuitive solution to this problem may be to model each relation R_j of an IS_i as a separate source IS_{ij}.

Example 2 *Using the scenario in Example 1, we now treat each relation as a separate IS. Assuming there is a data update ΔR_3 of R_3 in IS_2. To calculate the incremental change ΔV, the SWEEP mediator would sequentially need to first send down a MQ to IS_{11} (R_1) and then one to IS_{12} (R_2). The MQ sent to IS_{11} is defined by: $MQ_1 = \Delta R_3 \bowtie R_1$. When the query result MQR_1 comes back from IS_{11}, then the mediator tests if there is any concurrent data update and then sends query down to IS_{12}, which actually also corresponds to the same IS_1: $MQ_2 = MQR_1 \bowtie R_2$. If a data update ΔR_2 occurs before the MQ query is being processed in IS_{12}, the query result MQR_2 return: $MQR_2 = MQR_1 \bowtie (R_2 + \Delta R_2)$ would be incorrect. SWEEP detects the concurrent data update ΔR_2 and is able to apply local compensation to remove $MQR_1 \bowtie \Delta R_2$ to get the correct ΔV because MQR_1 and ΔR_2 are in DW.*

From Example 2, we notice that SWEEP can indeed support multi-relation ISs by treating each relation as a separate IS. However, this solution suffers from numerous shortcomings. First, the mediator would have to send down separate MQs to the same IS multiple times (one for each relation in it) for calculating a ΔV from one DU. This generates overhead in terms of network communication between the ISs and DW. In Example 2, the mediator sends MQ queries down to IS_1 twice to calculate ΔV, since IS_1 holds 2 relations. If there were 10 relations for each IS, then to handle one update, the DW in Example 2 has to send 10 remote sub-queries to and receives 10 query result messages from the IS as compared to one message exchange only in Example 1. Together with the added delay from IS query processing this network transfers, this would delay the refresh of the view extent of DW. So the DW has longer periods of being out-of-date. Second, each individual IS needs to receive, handle and process n (n representing the number of relations utilized in this IS) different queries instead of one single query. This places a burden on the IS, potentially affecting not only the handling of this one DU but also the response time of other users of this IS.

1.2 Our Approach

To overcome the limitation of the strawman solution described in Section 1.1 we instead propose a more efficient solution, called the Multi-Relation Encapsulation Wrapper, in short MRE Wrapper. The basic idea of MRE Wrapper is to treat an IS composed of multiple relations as one local view so that the DW will be aware of this one local view relation only instead of the redundancy relations for the IS. Hence, existing algorithms for DW maintenance would function unchanged within this environment once enhanced by our MRE Wrapper.

This implies that the wrapper will need to receive queries from the DW expressed against one relation, namely, the view relation modeling the content of the IS, and then translate this query down into one processable by the actual IS. Similarly, the wrapper will translate one update message for one relation into an update message with respect to the view relation of the IS. In order to calculate the effect of one data update on the whole IS without the threat of an infinite wait, the wrapper needs to adopt a local compensation strategy.

As we will demonstrate in this paper, our MRE Wrapper meets the following goals:

- **non-intrusive**: It encapsulates the update detection and query processing specific to one IS and thus does not require any modification to the existing processes and algorithms in a DW system. That is, the interface of the DW layer with the IS layer remains unchanged, and the fact that the view relation models many actual IS relations is transparent.
- **efficient**: It maintains all benefits of previous view maintenance solutions, while in addition offering improved performance to the overall process. Unlike the candidate solution described in Section 2, it preserves the property of [1] to do local and not remote compensation.
- **flexible**: It has limited requirements upon the underlying environment. In particular, the ISs can be semi-autonomous, i.e. they do not need to assist us with the DW maintenance process beyond reporting data updates or processing queries send down to them by the wrapper.
- **well-behaved**: It passes up to DW the view maintenance query results that incorporate the effects of all local concurrent data updates that take place while determining the query result. This MQR compensation is based on local correction techniques and thus does not require any infinite wait for refreshing DW.

Outline: In Section 2, the underlying DW model is given. Section 3 analyses the MRE Wrapper requirements. In Section 4, the MRE Wrapper architecture and algorithm are presented. Conclusions are discussed in Section 5.

2 The Data Warehouse Model Augmented with the MRE Wrapper

We assume a standard three-tier DW architecture. The environment is divided into three layers, the *data warehouse layer*, the *mediator layer* and the *information source layer with wrappers*. The three layers are respectively connected by a FIFO network. We also assume that the communication between wrapper and the DBMS of the IS where the wrapper is located is FIFO.

At the DW layer, the DW is materialized and directly responds to query requests by the users. At the middle layer, the mediator integrates the changes into the DW by merging the updates of the ISs with the data already present in the DW and resolving possible update anomalies. At the IS layer, the wrapper detects changes at its designated IS and propagates the changes to the upper layer.

There is a lot of work in the literature on the DW layer of concurrency control [5] and middle layer of view maintenance [10, 11, 1, 8]. In this paper, we focus on the design of the wrappers for such concurrent environments.

Table 1 shows the notation we use in this paper.

2.1 Requirements of the Mediator

The mediator is responsible for collecting messages from the corresponding wrappers at the ISs and maintaining the materialized view stored in the data warehouse. For every view located in the data warehouse, there is one mediator.

Notation	Meaning
AQ	Assembly query used by VM in DW.
MQ	Maintenance query from mediator to wrapper.
MQR	Query result of MQ.
LQ	Local query in wrapper to generate ΔIS.
LQR	Query result of LQ
LMQ	Localized maintenance query.
LMQR	Query result of LMQ.
V_IS_i	An local view definition stored in IS_i wrapper. Same as LQ.
ΔV_IS_i	ΔIS from IS_i. [1]
ΔR_i	A data update from relation R_i.
$\Delta IS(R_i)$	ΔIS generated from data update R_i.
ΔIS	Effect of a ΔR to the whole IS.

Table 1: Notation and Meaning

Our claim is now that the utilization of the proposed MRE Wrapper will allow us to plug in any existing incremental view maintenance algorithms without requiring any change to the mediator. The only thing we need is to add the module for initialization of the system, in particular to help to properly utilize the view query needed by the DW from the MRE Wrapper. More details about initialization are given in Section 3.3.

The general requirements of the mediator that can cooperate with the MRE Wrapper are:

1. The *mediator* is responsible for only one view definition in the data warehouse.
2. The *mediator* can maintain the DW in the distributed environment with multiple ISs.
3. The *mediator* handles the concurrent DUs in its view maintenance process.

SWEEP and PSWEEP meet all these requirements and thus work well with our MRE Wrapper [3].

3 Analysis of Requirements of the MRE Wrapper

Besides traditional wrapper functionality, the MRE Wrapper also supports multiple relations and handles concurrent local DUs. To support multiple relations in one IS, the MRE Wrapper stores one local view definition for each view of the DW. This view definition generated at the system initialization time will be used to calculates ΔIS for each ΔR. The MRE Wrapper does not actually materialize the local view, instead it directly calculate ΔIS by joining of ΔR and the underlying relations of that IS specified by the local view.

3.1 Black-box Analysis of the MRE Wrapper

If we treat the MRE Wrapper as a black-box, we can identify the following inputs, outputs, and function requirements.

- **Input:** Receives maintenance queries (MQs) from the mediator, query results (LMQRs and LQRs) and ΔR from its designated IS.

[1] Here, ΔV_IS_i, $\Delta IS(R_i)$ and ΔIS have same extent.

- **Output:** Sends ΔISs and MQRs to the mediator, forwards LMQ queries to the ISs used to process the MQ, sends LQ queries down to the IS relations to calculate the $\Delta IS(R_i)$ from ΔR_i.
- **Functions:**

  (1) Generates ΔIS for each ΔR_i with R_i being a relation in IS.

  (2) Processes the MQ and returns the MQR.

  (3) Ensures the correct order of returning messages. If ΔIS is sent to DW by the wrapper before (after) the MQR, then MQR is generated to (not) include the effects of ΔR.

3.2 White-box Analysis of the MRE Wrapper

Every IS will have a wrapper for every mediator of the DW that integrates data from two or more relations from this IS. In order to ensure the functionalities described in Section 3.1, we have the following implementation requirements:

1. **Local View Definition**. The calculation of ΔIS out of each ΔR is based on the local view definition specific to each IS established for the materialized view defined in the DW.
2. **Local Correction**. The calculation of ΔISs for any ΔR will be corrected in the wrapper by a local compensation (LC) technique. For each ΔR, there is one ΔIS generated and sent to the mediator.
3. **Single Transaction to Calculate MQR**. MQ is executed by the wrapper and MQR is returned to the mediator. MQR contains the effect of all concurrent ΔRs that happened at the IS during the execution.
4. **Order Reassignment to Ensure Correctness.** ΔIS and MQR will be sent in such an order that the later one will have the effects of the previous one incorporated.

The data update calculation has to utilize local compensation techniques, otherwise IS cannot report the ΔIS for a specific ΔR because of continuously happening concurrent ΔRs. Hence the IS wrapper can't process other queries due to waiting for this calculation of ΔIS. Then the whole maintenance process in the mediator would be blocked waiting for that specific IS. Therefore, the wrappers have to use the local correction techniques to calculate ΔIS.

3.3 System Initialization

To use the MRE Wrapper in the DW system, the initialization phase should decompose the DW view definition into local view definition for every IS wrapper and generate a assembly query at the DW to maintain the materialized view. Then it will use the local view to initialize the corresponding wrapper for each IS. The following operations are required during system initialization.

1. The user view definition at the DW will be decomposed into local view queries for each of the involved IS.
2. There is one assembly view based on the local IS views, which is stored in the mediator.
3. The DW system initializes the wrappers of informations sources by their respective local views.

This initialization process is described below using the same scenario as Example 1.

Example 3 *Table 2 shows the schema of the relations in each IS. A view is defined in the DW by Query 1.*

IS name	*Relation Name*	*Attribute Name*
IS_1	R_1	*(A, C)*
	R_2	*(D, E)*
IS_2	R_3	*(B, F)*

Table 2: Relation Structure

DW View Q1:

CREATE VIEW	*V AS*
SELECT	*A, B*
FROM	$IS_1.R_1$, $IS_1.R_2$, $IS_2.R_3$
WHERE	$IS_1.R_1.C = IS_1.R_2.D$ *AND*
	$IS_1.R_2.E = IS_2.R_3.F$

Figure 1: Data Warehouse View Definition.

During the initialization phase of the mediator, query Q1 will be decomposed into query Q2 for IS_1 and query Q3 for IS_2 to create the local views. As we can see from Figure 2, the local views only contain a subset of information of one IS as required by the DW view.

The mediator is based on the assembly query Q4 defined in Figure 3, which is used in place of the initial user provided query Q1 to maintain the DW. The assembly query Q4 uses sub-queries Q2 and Q3.

Local View Q2 of IS_1:

CREATE VIEW	*V_IS_1 AS*
SELECT	*A, E*
FROM	R_1, R_2
WHERE	$R_1.C = R_2.D$

Local View Q3 of IS_2:

CREATE VIEW	*V_IS_2 AS*
SELECT	*B, F*
FROM	R_3

Figure 2: Local Views at the Information Sources

4 Design of the MRE Wrapper Module

4.1 Architecture of the MRE Wrapper

Figure 4 shows the MRE Wrapper architecture. There are two data structures in the wrapper. The **Wrapper Message Queue (WMQ)** buffers all incoming data updates (ΔR) from relations and LMQ from the Localized Processor. The **Order Message Queue (OMQ)** buffers and reorders the output messages (MQR and ΔIS) to ensure the messages will be send to the mediator in the correct order. The MRE Wrapper is composed of five processes.

DW Assembled Query Q4:

CREATE VIEW	*V AS*
SELECT	*A, B*
FROM	*V_IS_1, V_IS_2*
WHERE	*V_IS_1.E = V_IS_2.F*

Figure 3: Assembly Query

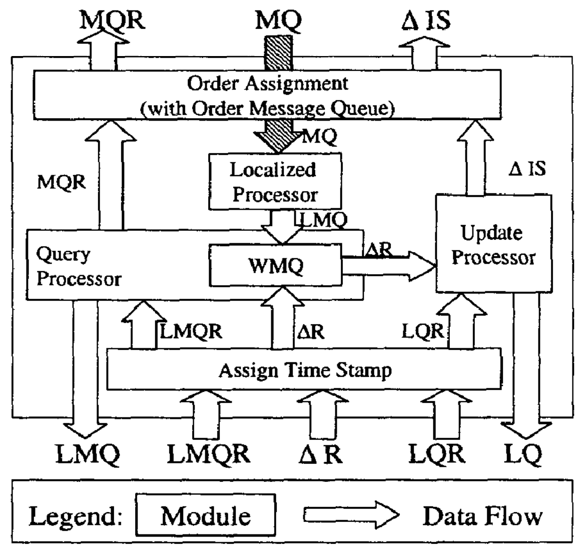

Figure 4: Architecture of the MRE Wrapper

The **Localized Processor** accepts MQ from the mediator. It localizes MQ in the sense of translating MQ into a query LMQ understood by the IS.

The **Assign Time Stamp** process assigns a local time stamp to all incoming messages, i.e, ΔR and LMQR from the underlying relations. The time stamps will be used by the *Query Processor* to ensure the sequential handling of incoming messages.

The **Update Processor** calculates ΔIS for ΔR. For this purpose, it first generates local queries (LQ) based on the local view definition stored in the wrapper at system initialization for a specific ΔR to produce ΔIS. Then it sequentially processes LQ to calculate ΔIS and fixes any concurrency update problem using a local compensation technique.

The **Query Processor** handles messages in the *Wrapper Message Queue* sequentially. There are two kinds of messages in WMQ: LMQ and ΔR. When the *Query Processor* gets a LMQ from WMQ, it will send the LMQ down to the IS relations and get the LMQR back within one transaction. The LMQR is forwarded to the *Order Assignment Processor.* When the *Query Processor* gets ΔR from WMQ, it will forward it to the *Update Process.*

The **Order Assignment** processor reorders the message (e.g., ΔIS and MQR) sequence. It sends MQR and ΔIS back to the mediator according to their time stamp to ensure the correct sending sequence.

Theorem 1 *As long as the wrapper sends ΔIS and MQR in the order that the latter one has the effect of all the previous data updates, the mediator can correctly maintain the DW.*

4.2 Calculating ΔIS under Concurrent DUs

As we stated before, the MRE Wrapper reports ΔIS for every ΔR. Because we do not actually materialize local views in the wrapper, the wrapper needs to send down a query LQ to the IS relations to calculate ΔIS for each ΔR and the $\Delta IS(R)$ should only have the effect of ΔR.

Theorem 2 *There will be no potential threat of an infinite waiting when the wrapper calculates ΔIS for a ΔR using a local compensation technique to handle concurrent DUs.*

The example below shows how the MRE Wrapper correctly calculates ΔIS from ΔR in a concurrent data update environment by using a local compensation technique. The system has been initialized as described in Section 3.3.

Example 4 *Assume there are DUs at IS_1. At time t1, ΔR_1 arrives at the wrapper from R_1. The wrapper calculates $\Delta IS(R_1)$ and reports it to the mediator. To calculate $\Delta IS(R_1)$, the wrapper sends down to R_2 the query $LQ = \Delta R_1 \bowtie R_2$. If a concurrent update ΔR_2 happens when $\Delta IS(R_1)$ is being calculated, the wrapper receives ΔR_2 from R_2 with time stamp t2 and gets the query result LQR back with time stamp t3. $LQR = \Delta R_1 \bowtie (R_2 + \Delta R_2) = (\Delta R_1 \bowtie R_2) + (\Delta R_1 \bowtie \Delta R_2)$. LQR has the effect of ΔR_2, which is incorrect. When the wrapper gets the LQR back, it checks their time stamp. From $t2 \leq t3$, the wrapper detects that a concurrent data update had occured. The wrapper knows the need to eliminate ΔR_2 from $\Delta IS(R_1)$. Although the wrapper does not store any view, it has the information of ΔR_1 and ΔR_2. To get the correct result, we can use a local compensation technique as for example defined in [1]. $LQR = LQR - \Delta R_1 \bowtie \Delta R_2$ Finally, the correct $\Delta IS(R_1) = LQR$ is sent to the mediator.*

4.3 Algorithm of the MRE Wrapper

Based on the previous description of the key features, we give the pseudo code of the MRE Wrapper module in Figure 5.

The *Query Processor* is used to process the query LMQ. Because the wrapper is only for one IS, the relations in the same IS are local and centralized. The LMQ can be sent to the DBMS of the IS and the DBMS processes LMQ and sends the query result LMQR back to wrapper in one transaction.

The *Update Processor* is invoked for every update received at the wrapper to generate ΔIS. It sequentially calculates ΔIS and erases any abnormal behavior by local correction techniques.

Below is an example of how the MRE Wrapper algorithm works and how it communicates with the mediator. We can see how the MRE Wrapper receives MQs query from mediator and executes the query and sends query result back.

Example 5 *Assume there is a data update ΔR_3 on relation R_3 of IS_2. The* Update Processor *of the wrapper of IS_2 will generate the query LQ based on Q3 to calculate $\Delta V\_IS_2(R_3)$. Then the wrapper reports the effect of ΔR_3, that is $\Delta V\_IS_2(R_3)$ to DW. After receiving $\Delta V\_IS_2(R_3)$ from IS_2, the mediator will generate maintenance query Q5 defined in Figure 6 with $\Delta V\_IS_2(R_3)$ and send the MQ to IS_1 to calculate ΔV.*

After the wrapper of IS_1 receives MQ Q5 from the mediator, the Localized Processor *merges the MQ query Q5 and the local view Q2 to generate LMQ (Figure 7). The* Assign Time

```
MODULE MRE Wrapper;
  CONSTANT
  GLOBAL DATA
    V: RELATION; /* Initialized to the local view */
    WrapperMessageQueue: QUEUE initially 0;
    OrderMessageQueue: QUEUE initially 0;

  PROCESS UpdateProcessor(ΔR: Relation; UpdateSource:
    INTEGER; TimeStamp: INTEGER): RELATION
  VAR
    ΔIS, TempIS: RELATION;
    j: INTEGER;
  BEGIN
    ΔIS = ΔR;
    /* Compute the left part of the ΔIS from ΔR */
    FOR (j = UpdateSource -1; j ≥ 1; j–) DO
      TempIS = ΔIS;
      SEND ΔIS to Source Relation i;
      RECEIVE ΔIS FROM Sour Relation i;
      IF ∃(ΔR, j, t) ∈ WrapperMessageQueue
        Then ΔIS = ΔIS - ΔR_j ⋈ TempIS;
      ENDIF
    ENDFOR;
    /* Compute the right part to the ΔIS from ΔR */
    FOR (j=UpdateSource+1; j ≤ n; j++) DO
      TempIS = ΔIS;
      SEND ΔIS to Source Relation i;
      RECEIVE ΔIS FROM Sour Relation i;
      IF ∃(ΔR, j, t) ∈ WrapperMessageQueue
        Then ΔIS = ΔIS - ΔR_j ⋈ TempIS;
      ENDIF
    ENDFOR;
    RETURN ΔIS;
    ENDAREA
  END UpdateProcessor;

  PROCESS AssignTimeStamp;
  VAR
    t: TIME; /* current system time at the IS */
  BEGIN
    LOOP
      RECEIVE Message FROM Relation i and LMQ FROM LocalizePro-
cess() as received order;
        t= getCurrentTime();
        APPEND (Message,i, t) TO WrapperMessageQueue;
    FOREVER;
  END AssignTimeStamp;


  PROCESS QueryProcessor;
  BEGIN
    WHILE WMQ not empty
      REMOVE a Message FROM WrapperMessageQueue;
      IF the Message is LMQ THEN
        SEND LMQ to Relations /*to calculated ΔV (LMQR)*/
        RECEIVE LMQR FROM Relations
        t= getCurrentTime();
        OrderAssignment(LMQR, t);
      ELSE /*Message is ΔR need calculate ΔIS */
        UpdateProcessor(Message.ΔR, i,Message.t)
        OrderAssignment(ΔIS);
      ENDIF
    ENDWHILE
  END QueryProcessor;

  PROCESS OrderAssignment(QueryResult,t);
  VAR
    r: INTEGER;
  BEGIN
    IF the QueryResult is LMQR THEN
      r = 0;
      WHILE WMQ is not empty
        IF ∃(ΔR, t') AND (t' ≤ t)
          r=1;
          IF (ΔR,t') is not in OMQ /*Order Message Queue*/
            APPEND (ΔR,t') TO OMQ;
          ENDIF
        ELSE
          BREAK;
        ENDIF
      END WHILE
      IF (r = 0)
        SEND LMQR TO Mediator;
      ELSE
        APPEND (LMQR,t) TO OMQ;
    ELSE /*Query Result is ΔIS*/
      SEND ΔIS(R) TO Mediator;
      IF ΔR is in OMQ
        DELETE ΔR FROM OMQ;
        WHILE head of OMQ is LMQR
          DELETE LMQR FROM OMQ;
          SEND LMQR TO Mediator;
        END WHILE
    ENDIF
  END OrderAssignment;

BEGIN /* Start MRE Wrapper Processes */
  StartProcess(AssignTimeStamp);
  StartProcess(QueryProcessor);
END MRE Wrapper Process
```

Figure 5: Pseudo Code of the MRE Wrapper Module

MQ Q5:

$$\begin{array}{ll} \textit{SELECT} & A,\ B \\ \textit{FROM} & V\_IS_1,\ \Delta V\_IS_2 \\ \textit{WHERE} & V\_IS_1.E = \Delta V\_IS_2.F \end{array}$$

Figure 6: Maintenance Query (MQ) Send to IS_1

Stamp *of the wrapper of* IS_1 *assigns a time stamp to this LMQ query and buffers in the WMQ. The* Query Process *gets LMQ from the WMQ. To execute the query LMQ Q6, the* Query Process *sends LMQ Q6 to* IS_1 *and gets the query result LMQR back* in one transaction.

Assuming there is a concurrent data update ΔR_2 *when LMQ Q6 is being executed, the wrapper will assign the local time to* ΔR_2 *and the* Update Processor *will generate* $\Delta IS_1(R_2)$*. The query result LMQR of LMQ Q6 will have the effect of* ΔR_2*. When* Order Assignment *receives the LMQR Q6, it knows that there is a concurrent data update* ΔR_2 *by checking the local time stamp scheme. So it buffers the LMQR in the QM until* $\Delta IS_1(R_2)$ *is also received by the* Order Assignment. *Then it returns* $\Delta IS_1(R_2)$ *followed by the query result LMQR that is to ensure the later one has incorporated the effect of the previous data updates .*

LMQ Q6:

$$\begin{array}{ll} \textit{SELECT} & A,\ B \\ \textit{FROM} & (\textit{SELECT } A,\ E \textit{ FROM } R_1,\ R_2 \\ & \quad \textit{WHERE } R_1.C = R_2.D)\ \textit{AS } V\_IS_1, \Delta V\_IS_2 \\ \textit{WHERE} & V\_IS_1.E = \Delta V\_IS_2.F \end{array}$$

Figure 7: Localized Maintenance Query (LMQ) in IS_1

The mediator will receive a $\Delta IS_1(R_2)$ *followed by the query result MQR (Q5) from* IS_1*.* ΔV *is calculated. The mediator knows that the MQR has the effect of* $\Delta IS_1(R_2)$ *by this receive-order and* $\Delta IS_1(R_2)$ *is a concurrent data updates. Hence the local compensation will erase the effect of the concurrent* $\Delta IS_1(R_2)$ *and DW is correctly updated.*

5 Conclusions

Incremental view maintenance algorithms [1, 8] only support one relation per information source, which is unrealistic in practice. In this work, we propose a MRE Wrapper to overcome this problem. The main features of the MRE Wrapper are: 1. Supports multiple relations per IS by treating such an IS as if it were a single relation from the DW point of view. 2. Is transparent to the DW system and thus can easily work with any incremental view maintenance algorithms. 3. Handles concurrent DUs at the IS using local compensation technology thus avoiding the infinite waiting problem. In summary, the MRE Wrapper maintains all advantages offered by existing algorithms in particular SWEEP and PSWEEP, while also achieving the additional desired features of being non-intrusive, efficient and flexible. We are currently in the process of enhancing our distributed data warehouse system EVE [7] by this wrapper technology.

Acknowledgments. We would like to thank all the DSRG members located at FL319 in the Fuller Labs.

References

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

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

[3] L. Ding, X. Zhang, and E. A. Rundensteiner. Enhancing Existing Incremental View Maintenance Algorithms Using the Multi-Relation-Encapsulation Wrapper. Technical Report WPI-CS-TR-99-23, Worcester Polytechnic Institute, Dept. of Computer Science, August 1999.

[4] A. Kawaguchi, D. F. Lieuwen, I. S. Mumick, and K. A. Ross. Implementing Incremental View Maintenance in Nested Data Models. In *Workshop on Database Programming Languages*, pages 202–221, 1997.

[5] S. Kulkarni and M. Mohania. Concurrent Maintenance of Views Using Multiple Versions. In *International Database Engineering and Application Symposium*, pages 254–258, 1999.

[6] M. K. Mohania, S. Konomi, and Y. Kambayashi. Incremental Maintenance of Materialized Views. In *Database and Expert Systems Applications (DEXA)*, pages 551–560, 1997.

[7] E. A. Rundensteiner, A. Koeller, X. Zhang, A. Lee, A. Nica, A. VanWyk, and Y. Li. Evolvable View Environment. In *Proceedings of SIGMOD'99 Demo Session*, pages 553–555, May/June 1999.

[8] X. Zhang, L. Ding, and E. A. Rundensteiner. PSWEEP: Parallel View Maintenance Under Concurrent Data Updates of Distributed Sources. Technical Report WPI-CS-TR-99-14, Worcester Polytechnic Institute, Dept. of Computer Science, May 1999.

[9] X. Zhang and E. A. Rundensteiner. The SDCC Framework for Integrating Existing Algorithms for Diverse Data Warehouse Maintenance Tasks. In *International Database Engineering and Application Symposium*, page 253, Montreal, Canada, August, 1999.

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

[11] Y. Zhuge, H. Garcia-Molina, and J. L. Wiener. The Strobe Algorithms for Multi-Source Warehouse Consistency. In *International Conference on Parallel and Distributed Information Systems*, pages 146–157, December 1996.