Parallel Consistency Maintenance of Materialized Views Using Referential Integrity Constraints in Data Warehouses

Jinho Kim[1], Byung-Suk Lee[1], Yang-Sae Moon[1], Soo-Ho Ok[2], and Wookey Lee[3]

[1] Dept. of Computer Science, Kangwon Nat'l University,
192-1 Hyoja Dong 2, Chunchon, Kangwon, Korea
{jhkim, bsdream, ysmoon}@kangwon.ac.kr
[2] Dept. of Computer Science, Kosin University,
149-1 Dongsam Dong, Youngdo Ku, Pusan, Korea
shok@kosin.ac.kr
[3] Dept. of Computer Science, Sungkyul University,
147-2 Anyang 8 Dong, Anyang, Kyungki, Korea
wook@sungkyul.edu

Abstract. Data warehouses can be considered as materialized views which maintain the online analytical information extracted from distributed data sources. When data sources are changed, materialized views should be maintained correspondingly to keep the consistency between data sources and materialized views. If a view is defined through joining several source relations, an update in one source relation invokes a set of join subqueries thus the view maintenance takes much time of processing. In this paper, we propose a view maintenance algorithm processing these join subqueries in parallel by using referential integrity constraints over source relations. A relation which has several foreign keys can be joined with referenced relations independently. The proposed algorithm processes these join operations in parallel then it merges their results. With the parallel processing, the algorithm can maintain materialized views efficiently. We show the superiority of the proposed algorithm using an analytical cost model.

1 Introduction

Data Warehouses (DW) are composed of materialized views which maintain online analytical information extracted from data sources located physically at different sites. These materialized views can be used to process users' OLAP queries efficiently without accessing data sources [1,2,3]. Whenever data sources are changed, the materialized views defined on the sources should be maintained to keep them up-to-date. This process is called view maintenance. Of view maintenance methods, incremental maintenance has been widely used, because it incrementally maintains views by using only the updated portion of each source relation [4,5].

Data sources for data warehouses can be stored in remote areas and they can be changed independently. It is difficult to maintain views consistently over the

A Min Tjoa and J. Trujillo (Eds.): DaWaK 2005, LNCS 3589, pp. 146–156, 2005.

changes of data sources. Thus we need a technique which maintains views as the same sequence of changes and the same state as data sources. We call it view consistency maintenance [3,4]. There have been several algorithms for view consistency maintenance, such as ECA [5], Strobe [6], SWEEP [7], and PVM [8]. The ECA algorithm was developed for view maintenance in single source site. The Strobe and SWEEP algorithms proposed a view consistency maintenance for join views on source relations distributed at multiple sites. The PVM suggested a parallel processing algorithm for view maintenance, which handles multiple updates occurred in source relations concurrently. When any change (i.e., insertion or deletion) occurs in a source relation, all of these algorithms has to process a sequence of join operations serially which combine the update with the tuples of other source relations. From the results of these join operations, we can get the values to modify views. Because each source relation can be stored in different sites, the join operations take much processing time. In this paper, we develop a parallel view maintenance algorithm, called PSWEEP/RI (Parallel SWEEP with Referential Integrity) which executes these join operations in parallel.

Let's suppose that a source relation has several referential integrity constraints and a materialized view is defined as joining it with other source relations through foreign keys. The join operations are independent from each other thus they can be executed in parallel to build the view. By using this property of referential integrity constraints, this paper proposes a parallel processing algorithm, PSWEEP/RI, for view consistency maintenance, which processes join operations for view maintenance in parallel. This can reduce the processing time for view maintenance. Furthermore, this can also lessen the problem that view maintenance cost increases linearly when the number of source relations does. The remainder of this paper is organized as follows. Section 2 describes the related works and the motivation of this paper. Section 3 presents the basic concepts of the proposed PSWEEP/RI algorithm. Section 4 analyzes the performance evaluation then, finally, Section 5 concludes this paper.

2 Related Works and Motivation

Materialized views in data warehouse are defined as join views on source relations distributed over several sites. When one of source relations is changed, the corresponding materialized views should be maintained to accommodate the source updates. For this view maintenance, data warehouse has to execute join operations of the changed source relation and all other source relations used to define the views.

For example, suppose there is a data warehouse view defined on four source relations shown in Figure 1. When some changes U_1 (i.e., ΔR_2) happen at the source relation r_2 and they are sent to data warehouse, the data warehouse has to execute the following view maintenance query Query1 to compute the information (ΔV) changing its view. In order to execute Query1, as shown in the Figure 1, the subqueries to join δR_2 and other source relations (R_1, R_3, and

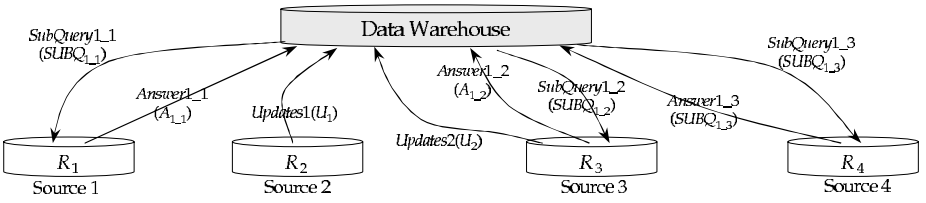


Fig. 1. View maintenance process over distributed sources

R_4) are sent in turn to the source sites and are executed [1,4] as follows. (Here, $Answer1\_1$ is the result of $SubQuery1\_1$).

$$Query1 = R_1 \bowtie R_2 \bowtie R_3 \bowtie R_4$$
$$SubQuery1\_1 = R_1 \bowtie R_2$$
$$SubQuery1\_2 = Answer1\_1 \bowtie R_3$$
$$\ldots \qquad \ldots \qquad \ldots$$

These source relations are stored in different sites each other and they can be changed independently at the same time. Thus it is very difficult to keep the consistency between materialized views and source relations. For example, when the source relation r_3 happens some changes U_2 (i.e., ΔR_3) before the $Answer1\_2$ is computed at the site, the answer of the $SubQuery1\_2$ can involve the state that the updates U_2 had already occurred, which is different from the state of source relation r_1. Thus the views may not be maintained consistently. Therefore, we need view maintenance techniques to guarantee the consistency between data warehouse views and distributed source relations. For these techniques, ECA, Strobe, SWEEP, and PVM algorithms have been proposed [5,6,7,8].

Zhuge et al. [5] introduced ECA algorithms ensuring the consistency of views for the cases that source relations are stored at a single source site. The Strobe algorithm [6] considers distributed data sources, requiring materialized views to contain a key for each of base relations and also requiring quiescence before installing any changes in materialized views. The SWEEP algorithms are introduced by executing serially view maintenance queries for updates in source relations as the order of their arrivals [7]. In the SWEEP algorithm, the join operations included in each view maintenance query are executed sequentially like Figure 1. It can incur much processing time when many resource relations are involved and/or source relation updates happen very frequently. In order to solve the problems of the SWEEP, the PVM algorithm [8] extends the SWEEP by invoking and parallelizing multiple threads for view maintenance, one thread for a view maintenance query. However, each thread executes the join operations of its query sequentially as same as the SWEEP does. Therefore, this paper proposes another approach for view consistency maintenance executing the join subqueries in parallel. By doing this, we can process efficiently view maintenance queries and we can also reduce the problem that view maintenance cost increases as linear as the number of source relations increases.

3 PSWEEP/RI Algorithm

This chapter describes the basic concepts of PSWEEP/RI which we propose. The algorithm processes join operations among source relations in parallel or it filters out view maintenance operation without processing any join among them [1].

3.1 Strategies in the SWEEP Algorithm

Let's suppose that there are referential integrity constraints on source relations and materialized views are defined as the joins among them. These referential integrity constraints can be represented by a graph as shown in the Figure 2. In the figure, ENROL relation refers to STUDENT, COURSE, and PROFESSOR relations. We call that it has *referring relationship*. When a relation refers to multiple relations, it is called *multiple referring*. Some relations such as STUDENT, COURSE, and PROFESSOR can be referred by others. We call they have *referred relationships*. When a relation is referred by several relations, it is called *multiple referred*.

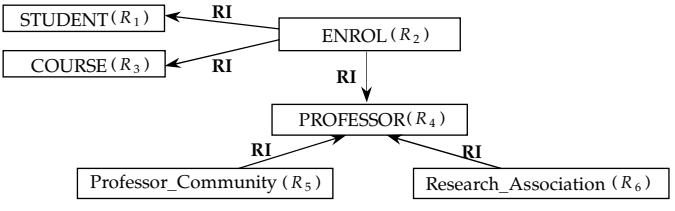


Fig. 2. Referential Integrity (RI) graph

When some changes(i.e., ΔR_2) occurs in a relation R_2 (i.e., ENTROL), in this example, data warehouse has to perform the following query to get the information for maintaining its views:

$$\Delta V = R_1 \bowtie \mathbf{\Delta R_2} \bowtie R_3 \bowtie R_4 \bowtie R_5 \bowtie R_6$$

In the existing SWEEP algorithm, this query is executed as follows: the join operations in the left-hand side of ΔR_2 are processed at first then the rest join operations in the right side of ΔR_2 are done. (Refer to the following procedure.) These join operations are executed in sequential order. In order to perform each join operation, furthermore, a subquery is sent to the site storing the relation to be joined. If the number of source relations defining a view increases, the number of join operations to execute increases thus the processing time for view maintenance increases too.

Left Sweep: $\Delta V_{left} = R_1 \bowtie \mathbf{\Delta R_2}$
Right Sweep: $\Delta V = \mathbf{\Delta V_{left}} \bowtie R_3 \bowtie R_4 \bowtie R_5 \bowtie R_6$

3.2 Maintenance Strategy for Multiple Referring Relation

Suppose some changes (i.e., ΔR_2) occur at a multiple *referring* relation (i.e., R_2) as shown in the above example. From RI graph described above section, we can identify that the relation R_2 refers to R_1, R_3, and R_4. In order to get the view maintenance information(ΔV), the join operations between ΔR_2 and R_1, R_3, or R_4 should be executed. To prove it, we present some lemmas and a theorem in the below. Here we assume that r_1, r_2, r_3, and r_4 relations are defined the schema R_1, R_2, R_3, and R_4 respectively, and $r_1 \rightarrow r_2$ indicates that relation r_1 refers to r_2.

Lemma 1. *Self-join of an identical relation r_1 using its primary key, produces the same relation as the original one. That is, $\pi_{R_1}(r_1 \ltimes r_1) = r_1$ holds.*

Lemma 2. (*commutative rule*) [9] *Join operation is commutative. That is, $r_1 \ltimes r_2 = r_2 \ltimes r_1$ holds.*

Lemma 3. (*associative rule*) [9] *Join operation is associative. That is, $(r_1 \ltimes r_2) \ltimes r_3 = r_1 \ltimes (r_2 \ltimes r_3)$ holds.*

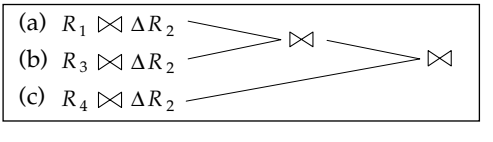
Theorem 1. (parallel join using referential integrity) *Let referential integrity constraints $r_1 \rightarrow r_2$, $r_1 \rightarrow r_3$, and $r_1 \rightarrow r_4$ exist and Δr_1 be some changes of a referring relation r_1. Then $\Delta r_1 \ltimes r_2 \ltimes r_3 \ltimes r_4 = \pi_{R_1,R_2,R_3,R_4}(\Delta r_1 \ltimes r_2) \ltimes (\Delta r_1 \ltimes r_3) \ltimes (\Delta r_1 \ltimes r_4)$ holds.*

PROOF: Let Δr_1 be r_1 for the simplicity, then the theorem is proved as follows.

$$
\begin{aligned}
&(r_1 \ltimes r_2) \ltimes (r_1 \ltimes r_3) \ltimes (r_1 \ltimes r_4) \\
&= (r_1 \ltimes r_2 \ltimes r_1) \ltimes r_3 \ltimes (r_1 \ltimes r_4) && \text{by Lemma 3} \\
&= (r_1 \ltimes r_1 \ltimes r_2) \ltimes r_3 \ltimes (r_1 \ltimes r_4) && \text{by Lemma 2} \\
&= (\pi_{R_1}(r_1 \ltimes r_1) \ltimes r_2) \ltimes r_3 \ltimes (r_1 \ltimes r_4) \\
&= (r_1 \ltimes r_2) \ltimes r_3 \ltimes (r_1 \ltimes r_4) && \text{by Lemma 1} \\
&= (r_1 \ltimes r_2) \ltimes (r_1 \ltimes r_4) \ltimes r_3 && \text{by Lemma 2} \\
&= (r_1 \ltimes r_2 \ltimes r_1) \ltimes r_4 \ltimes r_3 && \text{by Lemma 3} \\
&= (r_1 \ltimes r_1 \ltimes r_2) \ltimes r_4 \ltimes r_3 && \text{by Lemma 2} \\
&= (\pi_{R_1}(r_1 \ltimes r_1) \ltimes r_2) \ltimes r_4 \ltimes r_3 \\
&= (r_1 \ltimes r_2) \ltimes r_4 \ltimes r_3 && \text{by Lemma 1} \\
&= r_1 \ltimes r_2 \ltimes r_3 \ltimes r_4 && \text{by Lemma 2}
\end{aligned}
$$

□

By the above Theorem 1, we can observe that the result of sequential execution for the join operations between ΔR_1 and R_2, R_3, or R_4 is equal to the one of parallel execution for them. Therefore, we process the view maintenance query in parallel as follows.



That is, the join subqueries (a), (b), and (c) are executed in parallel then their results are merged in turn to get the final result. (For the simplicity of explanation, merging was represented by a join operation.)

3.3 Maintenance Strategy for Referred Relation

This section describes the strategies for changes on a referred relation, which is referred by other relations.

Theorem 2. (*filtering equijoin using referential integrity*) *Suppose that* $r_1 \rightarrow r_2$ *exist and the changes on the referred relation* r_2 *is* Δr_2. *Then* $r_1 \ltimes \Delta r_2 = \phi$, *where* Δr_2 *is the combination of insertions (notated by* $\blacktriangle r_2$*) and deletions (notated by* $\blacktriangledown r_2$*).*

Proof:

1) When some insertions occurred at r_2, if $r_1 \ltimes \blacktriangle r_2 \neq \phi$, then it means that the referring relation r_1 has tuples satisfying $r_1.FK = \blacktriangle r_2.PK$. These tuples violate the referential integrity constraint (i.e., $r_1 \rightarrow r_2$), it contradicts the definition of referential integrity.
2) When some deletions occurred at r_2,
   2.1) if there is any tuple in r_1 that the deleted tuples in $\blacktriangledown r_2$ refer to, it will be automatically deleted or changed into another value according to the options (i.e., cascade, set null, set default, or restrict) given to the referential integrity constraint. Therefore r_1 doesn't have any tuple to join with $\blacktriangledown r_2$ thus $r_1 \ltimes \blacktriangledown r_2 = \phi$. Otherwise,$\blacktriangledown r_2$ couldn't be performed.
   2.2) if there are no tuples in r_1 that refer to the tuples deleted from r_2, there are no tuples to join with $\blacktriangledown r_2$. Therefore $r_1 \ltimes \blacktriangledown r_2 = \phi$ must be true.

With both 1) and 2) above, $r_1 \ltimes \Delta r_2 = \phi$ is true. □

As proved in the Theorem 2, even though any tuple is inserted into or deleted a referring relation, the result of join operations for view maintenance queries is always null. In the proposed PSWEEP/RI algorithm, the maintenance queries for the changes of referred relations are filtered without executing any operations because they have no affection to the view updates.

4 Performance Evaluation

In this chapter, we present the results of performance evaluation comparing the proposed PSWEEP/RI and the previous SWEEP. For the evaluation, we use the materialized view example in Figure 2. In Section 4.1, we present an analytical cost model based on the example in Figure 2. In Sections 4.2∼4.4, we show the evaluation results computed by using the cost model.

4.1 PSWEEP/RI Cost Model

To design the analytical cost model for the example in Figure 2, we use the parameters described in Table 1. And, we also use the following assumptions to analyze the proposed algorithm.

- Since referential integrity constraints are enforced, the *index join* method will be used if a join has a referential integrity constraint.
- The *nested loop join* method will be used if a join has not any referential integrity constraint.

Table 1. Summary of parameters

Parameters	Definition/Meaning
$Ccom$	Bandwidth for transmitting data
$N(R_i)$	Number of tuples in relation R_i
$W(R_i)$	Average size of a tuple in relation R_i
$N(dR_i)$	Number of tuples in R_i delta relation
DW	Data warehouse
ΔR_i	Changed parts of the source relation R_i
$Send(\Delta R_i, DW)$	Cost for transmitting ΔR_i to DW
$Join(\Delta R_i, R_j)$	Cost for joining ΔR_i and R_j
B	Block size
k	Cost for depth first search in general case
br	Constant value for communication overhead

– Changes will be occurred in relation R_2 of the example, and let the changed parts of relation R_2 be ΔR_2.

We compute the total view maintenance cost of PSWEEP/RI by adding communication cost and join cost. Since communication operations in the same phase are able to be processed in parallel, the cost for these operations will be set to the maximum cost of them. Also, since join operations in the same phase are able to be processed in parallel too, the cost for these join operations will be set the maximum cost of them. The following equations show the analytical cost model of PSWEEP/RI for the example in Figure 2 (The cost model of the existing algorithm SWEEP can be derived as the similar way, but it is omitted due to space limitation.).

① $Send(\Delta R_2, DW) = (N(dR_2) \cdot W(R_2)/Ccom$ *Eq.* (1)

② $Send(\Delta R_{2(DW)}, R_1) = (N(dR_2) \cdot W(R_2)/Ccom$ *Eq.* (2)

② $Send(\Delta R_{2(DW)}, R_3) = (N(dR_2) \cdot W(R_2)/Ccom$ *Eq.* (3)

② $Send(\Delta R_{2(DW)}, R_4) = (N(dR_2) \cdot W(R_2)/Ccom$ *Eq.* (4)

③ $Join(\Delta R_2, R_1) = N(dR_2) \cdot \lceil \log_k N(R_1) + 1 \rceil \cdot br = X$ *Eq.* (5)

③ $Join(\Delta R_2, R_3) = N(dR_2) \cdot \lceil \log_k N(R_3) + 1 \rceil \cdot br = Y$ *Eq.* (6)

③ $Join(\Delta R_2, R_4) = N(dR_2) \cdot \lceil \log_k N(R_4) + 1 \rceil \cdot br = Z$ *Eq.* (7)

④ $Send(X, DW) = ((N(dR_2) \cdot (W(R_2) + W(R_1))) \cdot 8)/\ Ccom$ *Eq.* (8)

④ $Send(Y, DW) = ((N(dR_2) \cdot (W(R_2) + W(R_3))) \cdot 8)/\ Ccom$ *Eq.* (9)

④ $Send(Z, DW) = ((N(dR_2) \cdot (W(R_2) + W(R_4))) \cdot 8)/\ Ccom$ *Eq.* (10)

⑤ $Join(X, Y) = (\lceil (N(dR_2) \cdot (W(R_2) + W(R_1)))/B \rceil + \lceil (N(dR_2) \cdot (W(R_2) + W(R_1)))/B \rceil) \cdot br = M$ *Eq.* (11)

⑥ $Join(M, Z) = (\lceil (N(dR_2) \cdot (W(R_2) + W(R_1) + W(R_3)))/B \rceil + \lceil (N(dR_2) \cdot (W(R_2) + W(R_4)))/B \rceil) \cdot br = N$ *Eq.* (12)

⑦ $Send(N_{(DW)}, R_5) = \left(N(dR_2) \cdot \left(\sum_{i=1}^{4} W(R_i)\right) \cdot 8\right) /\ Ccom$ *Eq.* (13)

⑦ $Send(N_{(DW)}, R_6) = \left(N(dR_2) \cdot \left(\sum_{i=1}^{4} W(R_i)\right) \cdot 8\right) /\ Ccom$ *Eq.* (14)

⑧ $Join(N, R_5) = N(dR_2) \cdot \left(\lceil \log_k N(R_5) \rceil + \frac{N(R_5)}{N(R_4)}\right) \cdot br = I$ *Eq.* (15)

⑧ $Join(N, R_6) = N(dR_2) \cdot \left(\lceil \log_k N(R_6) \rceil + \frac{N(R_6)}{N(R_4)}\right) \cdot br = J$ *Eq.* (16)

⑨ $Send(I, DW) = \left(N(dR_2) \cdot \frac{N(R_5)}{N(R_4)} \cdot \left(\sum_{i=1}^{4} W(R_i) + W(R_5)\right) \cdot 8\right) /\ Ccom$ *Eq.* (17)

⑨ $Send(J, DW) = \left(N(dR_2) \cdot \frac{N(R_6)}{N(R_4)} \cdot \left(\sum_{i=1}^{4} W(R_i) + W(R_6)\right) \cdot 8\right) / \ Ccom$ *Eq.* (18)

⑩ $Join(I, J) = \left(\left\lceil \left(N(dR_2) \cdot \frac{N(R_5)}{N(R_4)} \cdot \left(\sum_{i=1}^{4} W(R_i) + W(R_5)\right)\right) / B \right\rceil + \left\lceil \left(N(dR_2) \cdot \frac{N(R_6)}{N(R_4)} \cdot \left(\sum_{i=1}^{4} W(R_i) + W(R_6)\right)\right) / B \right\rceil \right) \cdot br$ *Eq.* (19)

Total communication cost = *Eq.*(1) + *max*{*Eq.*(2), *Eq.*(3), *Eq.*(4)} + *max*{*Eq.*(8), *Eq.*(9), *Eq.*(10)} + *max*{*Eq.*(13), *Eq.*(14)} + *max*{*Eq.*(17), *Eq.*(18)}

Total maintenance cost(*join cost* + *communication cost*) = *Eq.*(1) + *max*{*Eq.*(2), *Eq.*(3), *Eq.*(4)} + *max*{*Eq.*(5), *Eq.*(6), *Eq.*(7)} + *max*{*Eq.*(8), *Eq.*(9), *Eq.*(10)} + *Eq.*(11) + *Eq.*(12) + *max*{*Eq.*(13), *Eq.*(14)} + *max*{*Eq.*(15), *Eq.*(16)} + *max*{*Eq.*(17), *Eq.*(18)} + *Eq.*(19)

4.2 Experimental Data and Environment

In analytical experiments, we compute the cost required in PSWEEP/RI and that in SWEEP by varying transmission bandwidth, relation size, and the number of tuples. We determine the database sizes on the basis of TPC-D, and adjust the other parameter values based on TPC-D database sizes. Table 2 shows the parameter values used in the experiments.

Table 2. Parameter values used in the analytical experiments

Parameters	Values
Ccom	100Kbps ∼ 100Mbps
$W(R_i)$	250 bytes
B	1,024 bytes
k	31
br	0.02

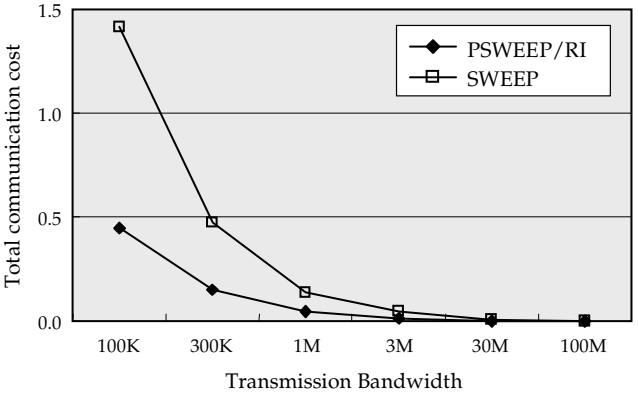


Fig. 3. Results of communication cost by varying the transmission bandwidth

4.3 Evaluation Results for Different Transmission Bandwidths

Figure 3 shows the changes of communication cost on different transmission bandwidths. The sizes of source relations have the same values determined in Section 4.2, and we set the number of updated tuples, $N(dR_i)$, to one and compute the costs by increasing the transmission bandwidth from 100Kbps to 100Mbps. As shown in Figure 3, we know that the communication cost of the proposed PSWEEP/RI is always less than that of the previous SWEEP. In particular, in the cases of lower bandwidths, PSWEEP/RI outperforms SWEEP significantly.

4.4 Evaluation Results for Different Relation Sizes

Figure 4 shows the changes of total communication cost on different numbers of updated tuples, i.e., on different $N(dR_2)$'s. We compute the cost by increasing $N(dR_2)$ from 1 to 32. As shown in the figure, we know that the communication cost of the PSWEEP/RI is always less than that of SWEEP due to the effect

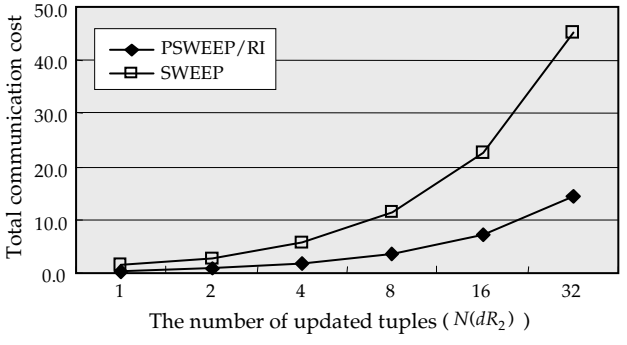


Fig. 4. Results of communication cost by varying the number of updated tuples

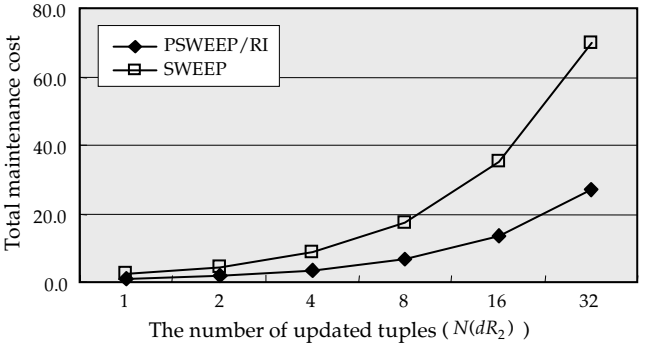


Fig. 5. Results of the total view maintenance cost by varying the number of updated tuples

of parallel processing. In particular, as $N(dR_2)$ increases, the cost difference becomes larger.

Next, Figure 5 shows the total view maintenance costs of two methods when we fix the bandwidth to 1Mbps and increase the number of updated tuples, $N(dR_2)$, from 1 to 32. As shown in the figure, in the case where the number of updated tuples is 32, which is only 0.002% of the source relation, the proposed PSWEEP/RI improves performance by 159% over the previous SWEEP.

5 Conclusions

In this paper, we proposed a view maintenance method which guarantees the consistency between data warehouse views and distributed source data efficiently by using parallel processing techniques. Existing algorithms like SWEEP execute sequentially join operations on source relations which are invoked to get the data for view maintenance. This sequential execution requires high processing cost for view maintenance. In order to reduce the processing cost, we proposed an algorithm, PSWEEP/RI, processing join operations on source relations stored different sites in parallel. We also designed a cost model to measure the processing cost of the proposed algorithm and evaluated its performance. From the experiments, we found that the proposed algorithm reduced both the communication cost and the total processing cost compared to existing methods.

The basic approach of the proposed algorithm is to take advantage of referential integrity properties in order to parallelize join subqueries involved in view maintenance. The join operations between a source relation (including multiple foreign keys) and its referenced relations (stored at different site) can be executed independently (i.e., in parallel). Then the join results can be merged together to obtain the final result for view maintenance. Furthermore, any changes on each referenced relation does not have any tuple in referenced relations. Thus view maintenance queries invoked by referenced relations can be filtered out without executing any join operations [1]. Because of these properties, he proposed algorithm reduced the cost of processing a sequence of join operations and the communication cost. Because of parallel processing, the algorithm can also reduce the problem that the view maintenance cost of existing methods increases in proportional to the number of source relations. For further researches, the proposed algorithm can be extended for join operations of source relations which doesn't employ referential integrity. This algorithm considered only join operations hence we also need to investigate parallel processing techniques for complex views involving other algebraic operations and aggregate functions.

Acknowledgements. This work was supported by the Korea Science and Engineering Foundation (KOSEF) through the Advanced Information Technology Research Center (AITrc).

References

1. Quass, D., Gupta, A., Mumick, I. S., and Widom, J., "Making Views Self-Maintainable for Data Warehousing," In *Proc. of Conf. on PDIS*, pp.158-169, 1996.
2. Colby, L., et al., "Supporting Multiple View Maintenance Polices," In *Proc. of ACM SIGMOD Conf.*, pp.405-416, 1997.
3. Gupta, A. and Mumick, I. S., "Maintenance of Materialized Views: Problems, Techniques, and Applications," *IEEE Data Engineering Bulletin*, Special Issue on Materialized views and Warehousing, Vol. 18, No. 2, pp.3-18, 1995.
4. Ross, K. A., Stivastava, D., and Sudarshan, S., "Materialized View Maintenance and Integrity Constraint Checking: Trading Space for Time," In *Proc. of Proc. of ACM SIGMOD Conf.*, pp.447-458, 1996.
5. Zhuge, Y., et al., "View Maintenance in a Warehousing Environment," In *Proc. of Proc. of ACM SIGMOD Conf.*, pp.316-327, 1995.
6. Zhuge, Y., Garcia-Molina, H., and Wiener, J. L., "The Strobe Algorithms for Multi-Source Warehouse Consistency," In *Proc. of Conf. on PDIS*, pp.146-157, 1996.
7. Agrawal, D., El Abbadi, A., Singh, A., and Yurek, T., "Efficient View Maintenance at Data Warehouses," In *Proc. of Proc. of ACM SIGMOD Conf.*, pp.417-427, 1997.
8. Zhang, X., Ding, L., and Rundensteiner, E. A., "Parallel Multisource View Maintenance," *The VLDB Journal*, Vol. 13, No. 1, pp.22-48, 2004.
9. Navathe, S. and Elmasri, R. Fundamentals of Database Systems, 4th ed., Addison Wesley, 2004.