Distributed Database Replicator Without Locking Base Relations

Wookey Lee, Sukho Kang, Jooseok Park*

Dept. of Computer Science, Sungkyul Univ., Anyang, Korea
wook@hana.sungkyul.ac.kr
Dept. of I.E. Seoul National Univ., Seoul, Korea
* Dept. of Business and Administration, Kyunghee Univ., Seoul, Korea

Abstract

A replication server is considered to be one of the most effective tools to cope with the problems that may be caused by the complex data replications in distributed database systems. In the distributed environment, locking a table is inevitable and it is the main reason to coerce the system practically. This paper presents an Asynchronous Replicator Scheme (ARS) that basically utilizes the system log as files named differential files to refresh the distributed data files with complicated queries, and that it prevents (normally, huge) base tables from being locked. We take join operations as the complicated queries, not only because the join operation covers almost all the operations, but also because it is one of the most time-consuming and data intensive operations in query processings.

1. Introduction

A data replicator represent a cost effective alternative for replicated data in distributed databases [Gold94, Wook96]. They are useful when users' applications may approve asynchronous but huge amount of data, such as a data warehouse, which a data replicator[1] can support. The simplest way to update the materialized views is to re-execute the view definition, but this causes unnecessary locks of tables and unacceptable communication costs. Herein, we propose the differential method. It does not reflect entire base tables, but only changed portions, by using the log as a differential file.

In this paper, we advance two basic principles to design a join scheme in a distributed environment.

(1) To minimize base table locks in order to make the distributed database systems more practical. The implication is that differential files will be primarily used instead of base tables.

(2) To reduce remote accesses to base tables, hence optimal reduction of data to be transferred to the relevant sites.

2. The Data Replicator Scheme

The notion of $\mathcal{R}$ is a relational schema that has a set of database relations and a relation R is an instance over $\mathcal{R}$. A base relation $R_j^u(TID^u, A_1,...,A_n, TS^u)$ has a name u and is located at site $j \in \Omega$, for $\Omega = \{1, 2, ..., n\}$ means the set of site index, where $A_1,...,A_r$ are attributes, and the TID^u means a tuple identifier of the base relation. TS^u is the time-stamp of the tuple and is recorded by the committed transactions.

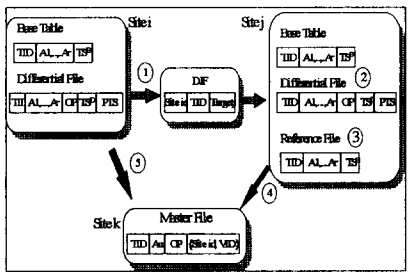
Views are materialized in various sites for the distributed environments, and some of them are selection views, some are selection-projection views, and others are join views (*SPJ views*). The schema of a view is defined as $V_i^t(VID, i, PRD_i, A_u, LU_i, NU_i)$ where VID is view identifier, $i \in \Omega$ is the site where the view is stored (the view name t and site identifier i can be omitted when the site has no other views with the relations); PRD_i is the predicate of the view definition; A_u means a set of attributes of the view; LU_i is the last update time; NU_i is the next update time.

A differential file is used to refresh materialized views and replicas, and has the following schema: $DF_j^u(TID^{ud}, A_1,...,A_r, OP, TS^{ud}, PT^{ud})$. The TID^{ud} is supposed to be the same as TID^u. We set it as TID^u or in short, TID. The OP indicates the type of operations done for each tuple; One of three codes: *'ins'*, or *'del'*, or $'del_m'$ and $'ins_m'$ in series, where *'ins'* means insertion, *'del'* deletion, and a

modification is described as 'del_m' and 'ins_m' in series with the same time-stamp (TS^u). TS^{ud} is the time the differential tuple was appended (The superscript d means differential file, here we assume that TS^{ud} is equal to TS^u). PT^{ud} is the previous value of TS^u having the same *TID*; if a tuple is newly inserted, the value of PT^{ud} is equal to Nil. Then the changed tuples (by committed transactions) and their operation codes (*OP*) with each time-stamps are recorded sequentially in the differential files.

Rather than sending the entire differential files, they may be compressed through the reduction procedure (as explained below). The reduction procedure and multiple query optimization techniques are addressed in [*Blak86, Hans87, SegP89, Wook96*].

The compressed tuples that passed the reduction procedure are pipelined to a procedure that is appended to a Master File having the following schema: *MF(TID, A_u, OP^v, {Site_id, VID})*, where OP^v indicates the type of updates to be done for each of the views on the list *{Site_id, VID}*, and the superscript v means that it will be one of the *OP*'s among relevant differential tuples; A_u represents a set of attributes. When OP^v is 'del', then it will be null (since the remote views need only a *TID* for a deletion); When OP^v is 'ins', then it will become the inserted data item of A_u; In case of modification, then A_u will be its new values.



[figure 4-1] ARS Join Scheme

When the relevant tuples to be joined will be collected at the site where the join operations are to be performed. If the join operations are made by various Master File tuples sent from different sites, it is difficult to manage these tuples as one relation, since the sizes of these tuples may differ. Therefore, we prescribe a new file called *DJF* (Differential Join File). The schema of *DJF* is as follows: *DJF (Site_ID, TID, A_u)*, where the *Site_ID* is a unique identifier of the site where the relevant relation is located, and A_u is the attribute that is used in join predicates, and it has pointers connecting the attributes of differential tuples used in materialized views.

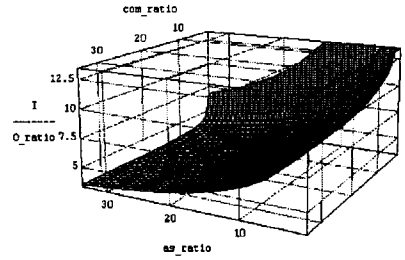
In the previous example, it is assumed that TID^p is the primary key of R_i^s and is the foreign key of R_j^p. The DJF_i can be created in no other site than site i, since we assume that relation *S* and *P* have one many relationship. (But generally speaking, it is up to the query optimizer in *DBMS*).
The changes of relation R_i^s means that the tuples of DF_i^s can be classified according to *OP's*, such as insertions, deletions, modifications, and unchanged.

5. Performance Analyses

The following values are assigned to the parameters for analysis. The communication speed varied between 10,000bps and 100,000,000bps. The screening factors(α_s) varied between 0.01 and 1.0; Here $\alpha_s = 1.0$ means that there is no screening and $\alpha_s = 0.0$ means 100% screening; *i.e.*, no tuple will be sent to other sites. We assumed that the size of a page(B) is 4000 bytes, the width of experimental Tables(W_{Ri} *and* W_{Rj}) 200 bytes, the size of B* tree(W_B) 8 bytes, input and output cost (C_{IO}) are 25 ms/block, and the sizes of *ARS* that inserted tuples(*Wins*) will be 200 bytes, deleted tuples(*Wdel*) 8 bytes, and the updated tuples (*Wmod*) 100 bytes, respectively.

Given the above values and varying the number of differential tuples, we can calculate the total cost of each algorithm. Varying the number of differential tuples and transmission rate, we suggest the total cost ratio being between semijoin algorithm and *ARS* scheme.

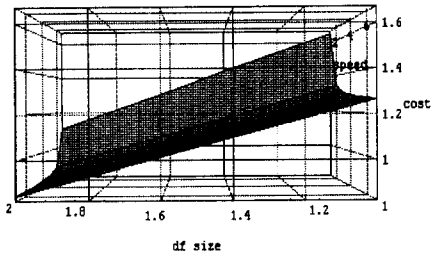
[*figure 5-1*] shows the total cost ratio of semi-join and *ARS* scheme three-dimensionally that means the total cost ratio at first is strongly dependent both on the screening factor and the communication speed, but the communication speed can be considered as a constant over 1M bps.



[Figure 5-1] Three dimensional results of total cost ratio.

[*figure 5-2*] shows that the size of the differntial file is a critical factor for distributed view updates in ultra-high communication network environments, and that the size of the differential file can be agreeable to up to about 180% of base Tables. We can conclude that the differential file scheme, that is to say the *ARS* scheme, is superior to the semijoin

scheme as long as the size of the differential file is up to 180% of the base Table for a very high communication speed *(say, 100M bps)*.



[Figure 5-2]The DF size aspect of total cost ratio

6. Summary

In this paper we proposed a data replicator scheme with a new algorithm, *ARS*, to efficiently support updating join materialized views in a distributed environment. An efficient join scheme should be devised not only because it can cover almost all operations in relational database systems but also because it is one of the most time-consuming and data intensive operations especially for a distributed environment. The performance analyses indicate that the total cost of this algorithm is closely dependent on the number of differential tuples, the screen factor, and the transmission rate. As these factors are reduced, so does the total cost. We proposed, therefore, a more effective tuple reduction procedure, since the other two parameters are strong system-dependent. The ratio of total cost to the transmission cost in algorithm *ARS* is much smaller than the ratio in the base relation scheme. In immediate update situations, the value of such a sophisticated reduction procedure is less emphatic, but advantages such as preventing the system from locking the whole base relations are still inact. A noteworthing fact is that the differential file scheme, i.e., the *ARS* scheme, is superior to the base table schemes as long as the differential file is up to 180% the size of the base table.

References

[*Adib80*] Adiba, M.E. and Lindsay, B., Database Snapshots, *Proc. of the 6th International Conference on Very Large Databases*, Montreal, Canada, Oct. 1980, pp. 86-91.

[*Alur95*] Alur, N., "Data Replication Techniques, Tools, and Case Studies", *in DB/PRDO '95 Conf. Material Thursday*, May 1995, pp. 83-102.

[*Blak86*] Blakeley, J. A., Larson, P. and Tompa, F. W., "Efficiently updating materialized views," *in Proc. ACM-SIGMOD Conf. Management of Data*, Washington D.C., May 1986

[*Gold94*] Goldring, R., "A Discussion of Relational Database Replication Technology," *InfoDB*, Spring 1994.

[*Onil95*] O'neil, P. and Graefe, G. "Multi-Table Jions Through Bitmapped Join Indices," *ACM SIGMOD Record*, Vol. 24, No. 3, Sept. 1995.

[*Rous91*] Roussopoilos, N., "An Incremental Access Method for ViewCashe: Concepts, Algorithms, and Cost Analysis, *ACM Transactions on Database Systems*, Vol. 16, No. 3, Sept. 1991.

[*SegP89*] Segev, A. and Park, J., "Updating Distributed Materialized Views," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 1 , no. 2, June 1989.

[*Vald87*] Valduriez, P. "Join Indices," *ACM Transactions on Database Systems*, Vol. 12, No. 2, June 1987.

[*Wook96*] Wookey, L., Jooseok, P., and Sukho, K., "Replication Server Scheme in Distributed Database Systems," *Proceedings of APDSI conference*, Vol. 3, 1996, pp. 1275-1282.