Transaction Acceleration in Secure Database Systems[1]

Ramzi A. Haraty and Roula C. Fany
Lebanese American University
Beirut, Lebanon
Email: rharaty@beirut.lau.edu.lb, rolafany@sodetel.net.lb

Abstract

Query acceleration and optimization continues to capture a great deal of attention because data in networked systems is distributed to many sites and data transfer is a necessity. Query optimization studies efficient techniques to minimize the cost and amount of data transferred. However, in multilevel secure systems, not only the amount of data is important but also the classification and flow of this data from and to specific sites. Multilevel secure systems are distributed systems where each site contains data categorized by security levels, which vary from unclassified to top secret. Each site cannot store data with higher security level. Some research has been done in this area. In this paper, an algorithm is presented to accelerate secured queries and the results are compared to other methods.

Keywords:
Multilevel Secure Databases, Joins, and Query Acceleration.

1.0 Introduction

The distribution of data over many sites has created new challenges and problems to solve in order to access information accurately, confidentially, and efficiently. Security of data in multilevel systems is of paramount importance. Many algorithms have proposed to store data securely and tried to suggest new techniques to control the mode of access privileges of users to data, hence preventing any unauthorized disclosure of information [3][6].

Among those techniques, we name Air Force Summer Study [6] that deals with classification of data. The basic idea is that data is classified according to certain security levels that may range from: unclassified - classified - secret - top secret. Each level is stored separately and in case of a distributed system, each site may store one specific level. The main restrictions to respect are that a user is not allowed to view information with higher security level and is allowed only to modify data at her/his level.

In this work, we propose a method that processes data securely and reduces the query response time of transactions in multilevel secure systems. We compare our results to other methods.

This rest of this paper is organized as follows: Section 2 gives an overview of the basic concepts. Section 3 presents our suggested algorithm. Section 4 shows an example of the calculations done. Section 5 presents the experimental results obtained when compared to a join without any acceleration. Section 6 concludes the paper and presents the future work to be done.

2.0 Basic Concepts

A MultiLevel Secure DataBase System (MLS/DBS) is a collection of users and data objects or relations [2]. Users are assigned different classification levels and data objects are assigned different sensitivity levels. Data are physically distributed and stored in separate databases according to sensitivity level with each relation storing only tuples with the same sensitivity level. It is the responsibility of the MLS/DBS to ensure that database users access only those data items for which they have been granted a clearance. This architecture is fairly secure since data are segregated and separated. However, performance overhead associated with multilevel transactions is a major disadvantage.

In order to prevent illegal disclosure of information, the flow of data should always go from lower security levels to higher security levels. Thus, traditional data retrieval mechanisms have to modified and, therefore, potentially become more complex.

The straightforward or unoptimized solution to query processing may ensure that confidentiality of data is maintained but would result in slow and inefficient queries while increasing the traffic on the network. Query optimization aims at minimizing unnecessary and redundant transfers by reducing data before shipment and then choosing a specific order of data flow between sites. The traditional method used in query processing is a three-phased approach that consists of the following:

- local processing to filter unnecessary data,

[1] Proceedings of the 13th National Conference on Computer Applications in Industry and Engineering (CAINE-2000). Honolulu, Hawaii, USA. November 2000.

- semi-join reduction involving shipment of data from one site to another, and
- final assembly at the destination site.

Using the above method, we notice that the flow of data is dependent on the maximum gain to be achieved by reducing the cost of transfer. Because this method is intended for medium-level secured systems, some optimization issues will be sacrificed for security purposes in order to prevent the disclosure of confidential information. So our algorithm will have as main target to prevent the flow of data from higher to lower security levels even if query processing will not be the most optimal in terms of amount of data transferred.

However, it is important to note that we will transfer back from higher level to lower level secured sites a bit vector representing the tuples of the lower level site that matched and that will be transferred to the destination site assuming that no confidential information is hidden or packed with this bit vector transmitted. This is acceptable since in multilevel secure database systems, the join attributes tend to be classified at a lower security level than the rest of the data in the database.

3.0 The RR-General Algorithm

Our query optimization algorithm will transfer data from low-level secured sites to high-level secured sites in order to reduce the data that needs to be shipped. During cost calculations, the transmission cost will be computed as a linear function of the size of the data. Schedule selectivity is calculated as a product of selectivities of all the attributes in the schedule. A selectivity of an attribute is defined as the number of distinct values divided by the number of possible values of the attribute [1][5].

We call our algorithm the RR-General algorithm, and it consists of the following steps:

1. Perform all initial local processing.
2. Set the cumulative selectivity to 1.
3. For each relation R do the following:
   i - Call the FORWARD_RR_GENERAL procedure.
   ii - Call the BACKWARD_RR_GENERAL procedure.
4. Ship resulting data to destination site.

3.1 Procedure FORWARD_RR_GENERAL

1. Order relations R_i such that
   $S_1 \leq S_2 \leq \ldots \leq S_m$
   where S_i is the security level of R_i

For each joining attribute of the current relation:

2. Transfer the joining attribute to the next relation by multiplying its size by the cumulative selectivity.

3. Let the cumulative selectivity = cumulative selectivity * selectivity of the joining attribute transmitted.
4. Choose the transmission of the joining attribute with the minimum cost.
5. Build a list, L, where $L_{R_i R_{i+1} j}$ is set to 1 when transmission was done from R_i to R_{i+1} on join attribute *j*.
6. When calculating transmission cost,
   If $L_{R_i R_{i+1} j} = 1$ then
   cost = 0
   Else
   cost = $C_0 + C_1 * b_{ik}$
   where :
   $C_0 + C_1 * b_{ik}$ is the linear function of transmission cost that is equal to the fixed cost per byte transmitted (C_1) multiplied by the size in bytes of the join attribute projected. This is the usual cost of a semi-join known as the forward cost.
7. The PERF bit vector is built and assigned the following cost:
   $(b_{ik} * b_{(i+1)k})/8$ is the backward cost that is the cost of transmitting back to R_i the bit vector consisting of only matching values of the corresponding attribute. For simplicity of this equation, we are considering attribute *k* of width 1 byte. This bit vector is sent back to R_i to be stored if R_i has a higher security level than R_{i+1} or else, it will be stored in R_{i+1} and transferred to R_i only when another join is needed between those two relations. In this case and for security purposes, we will not transmit for this moment this bit vector but we will store at the higher level site until the second phase: BACKWARD_RR_GENERAL.

3.2 Procedure BACKWARD_RR_GENERAL

1. Order relations R_i such that
   $S_m \geq \ldots \geq S_2 \geq S_1$
   where S_i is the security level of R_i
   For each relation, do the following:
2. Transfer the reduced relation to the destination site using the cost equations described above.
3. Transmit to the lower level site the bit vector in order to reduce it and send the reduced relation to the destination site.
   The total cost, will be the sum of all the above costs.

As it can be seen, the RR-GENERAL algorithm does not ship all relations to the destination site as the unoptimized method does, but it tries to reduce the relations before shipment to the query site. This reduction is limited by the security considerations, meaning that it does not provide the most optimal schedule but the safest one.

This reduction is ensured by the transmission of the PERF bit vector back to the original site in order to reduce its tuples for final shipment. This method will add a little overhead on the transmission cost, but this overhead is negligible compared to the gain obtained by the reduction effect. We note that the reduction effect of the algorithm is proportional to the width of the attributes used. In section 5, we show results from different width selections to clarify this issue.

4.0 A Comparative Example

Consider an AIRCRAFT database that describes a database for aircraft supply system. The database consists of the following relations:

1. **PARTS** (P#, PNAME): This relation identifies the different parts of a plane. It is stored at the unclassified level.
2. **ON_ORDER** (S#, P#, QTY): This relation identifies the supplier number for each part of the aircraft and the corresponding quantity on order. This relation is stored at the confidential level.
3. **S_P_J** (S#, P#, J#): This relation contains for each job number, the part numbers and from which suppliers they are. S_P_J is stored at the secret level.
4. **SUPPLIERS** (S#, SNAME): This relation identifies the different suppliers. It is stored at the top-secret level.

Also consider the following query: List the product number, name, supplier name and total quantity for all parts if the aircraft that are currently on order from suppliers who supply that part to jobs 1 or 2.

The two joining attributes are: P# and S#. The cost function to be used is: C(X) = 20 + X. It is a linear function in the form of y= aX + b where:

i- Cost added per byte transmitted.
ii- Fixed cost dependent on the network used. In this example b is taken as 20.

The corresponding size and selectivity relations are given in the following figure:

Ri	|Ri|	Si	di1 = P#		di2 = S#	
			bi1	ρi1	bi2	ρi2
R1	70	1000	400	0.9	100	0.9
R2	140	2000	400	0.9	450	0.9
R3	120	3000	900	0.9	-	-
R4	50	1000	-	-	75	0.2

Figure 1. Relations Description

For each relation we have as given:
|Ri|: cardinality of the relation (number of tuples).
Si : size of the relation in bytes.
dii : join attribute.
bii : for each joining attribute, the size, in bytes, of the column in the corresponding relation.
ρii : for each joining attribute, the corresponding selectivity.

Applying algorithm RR-GENERAL to this query, the following results are obtained using both procedures FORWARD_RR_GENERAL and BACKWARD_RR_GENERAL.

Using the procedure FORWARD_RR_GENERAL, the following ordering is produced:

$$R_1 - R_2 - R_3 - R_4$$

So the flow of data should always go from R_1 to R_4.

R1:
Cost1 = C(400)
= 420
Cost2 = C(100)
= 120
We choose Cost2 = 120
PERF (at R2) = 100 * 0.9 / 8
= 11.25
R2:
Cost = C(400 * 0.9)
= 380
PERF (at R3) = 80 * 0.9 / 8
= 9

Using the second procedure of the algorithm BACKWARD_RR_GENERAL, we get the following ordering:

$$R_4 - R_3 - R_2 - R_1$$

R4:
Cost = C(1000)
= 1020
R3:
Cost = C(3000 * 0.81)
= 2450
R2:
PERF = C(9)
= 29
Cost = C(2000 * 0.81) + 29
= 1669
R1:
PERF = C(11.25)
= 31.25
Cost = C(1000 * 0.81) + 31.25
= 861.25
Total cost = 6500.25

Using the unoptimized method we would get: 7080. Therefore, our contribution is: (7080 – 6500.25) / 7080 = 8.19% where contribution is equal to the initial time - enhanced time divided by the initial time. In our case the initial time is unoptmized time and the enhanced time is RR time.

5.0 Experimental Results

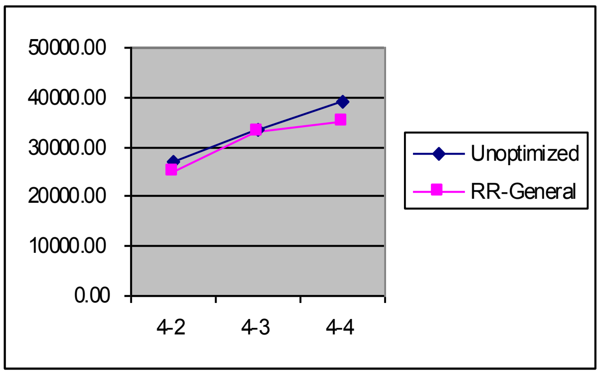
Different scenarios were conceived in order to evaluate the performance of the different algorithms and for each scenario programs were run 700 times.

Note that all programs were developed using Visual C++ 6.0 under Windows 98. Experiments were conducted on a Pentium V PC with 64 MB RAM.

5.1 Scenario 1

In this scenario the attribute width is taken as 1 byte for all attributes. The Type field in the table below indicates the number of tables joined and the maximum number of joining attributes. Graphically, the results are represented as follows:

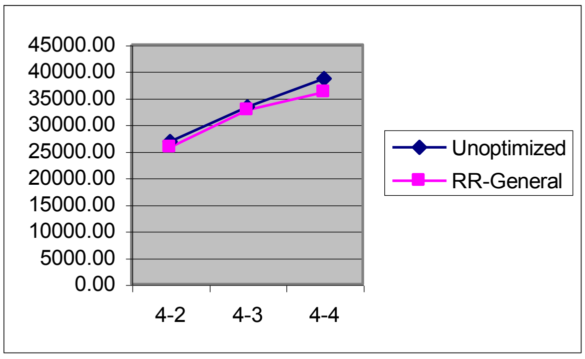
Type	Unoptimized	RR-General	Gain
4-2	26828.55	25192.84	6.09%
4-3	33498.00	32891.93	1.80%
4-4	39050.00	35259.15	9.70%



5.2 Scenario 2

In this scenario the attribute width is taken as 5 bytes for all attributes. Graphically, the results are represented as follows:

Type	Unoptimized	RR-General	Gain
4-2	26795.00	25682.93	4.10%
4-3	33259.86	32694.41	1.70%
4-4	38566.57	36035.22	6.50%



We used different scenarios in order to study the performance of the algorithms from different perspectives. For each scenario, we compared the performance of the algorithm with respect to the unoptimized solution. Using different scenarios we studied better the behavior of all algorithms under a variety of circumstances.

6.0 Conclusion and Future Work

In this paper, an algorithm using semi-joins was presented as our contribution to the query optimization problem for multilevel secured databases. Experimental results confirmed our expectations by showing an enhancement over the unoptimized method. However, based on the fact that during the query processing, data in the relations should not be updated without updating the list accordingly and because not much work has been done until now to deal with this problem, we view RR-General algorithm as a good solution for distributed query optimization for multilevel secured databases that can be adapted for huge, static warehouses where data is not changed very frequently.

References

[1] R. Fany, PERF Solutions For Distributed Query Optimization, Thesis, September 1999.

[2] D. E. Bell and L. J. LaPadula, Secure Computer Systems, The Mitre Corporation. March 1976.

[3] R. Haraty, “Concurrency Control and Query Processing in Multilevel Secure Kernelized Databases,” Proceedings of the Symposium on Applied Computing. Phoenix, AZ. April 1994.

[4] Alan R. Hevner, O. Qi Wu and S. Bing Yao, “Query Optimization on Local Are Networks,” ACM Transactions on Office Information Vol. 3, No. 1. January 1985.

[5] Zhe Li and K. A. Ross, “Fast Joins Using Join Indices,” VLDB Journal, Vol. 8, No. 1. 1999.

[6] Multilevel Data Management Security, Committee on Multilevel Data Management Security, Air Force Studies Board. National Research Council. Washington, DC. 1983.