Accelerating Queries in Multilevel Secure Database Systems[1]

Ramzi A. Haraty[1] and Roula C. Fany[2]

[1] Lebanese American University, P.O. Box 13-5053
Beirut, Lebanon
rharaty@beirut.lau.edu.lb
[2] Beirut Ryad Bank, Ryad Soloh Street, Beirut Ryad Bank Building, Beirut, Lebanon,
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 results are compared to other method.

1 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 accessed 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][7].

Among those techniques, we name Air Force Summer Study [6] that deals with classification of data. Hence, 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 stores 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 and her/his level.

In this work, we propose a method that will reduce the query response time of transactions in multilevel secure systems and compare our results to other methods.

This rest of this paper is organized as follows. Section 2 gives an overview of the basic concepts along with the assumptions taken. 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 Basic Concepts and Assumptions

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

[1] Proceedings of the 10th International Conference on Computing and Information ICCI '2000, Kuwait, November 18-21, 2000 (proceedings not published yet).

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 would ensure the 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,
- 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. For this purpose, we will assume at this point that the secure data to protect does not reside in the joining attributes that link between relations.

In our work, we also assume data transfer is done via secure communication channels that ensure the encryption of data while being transferred, so no interference can be done during this process.

3 The RR Algorithm

Our query optimization algorithm uses as basis, the AHY General Total Time algorithm, which was presented by Apers, Hevner and Yao [1][4]. Throughout this algorithm, different schedules are built and compared until the most optimal one is chosen. 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 [5].

In our RR algorithm, the following steps are performed:

1. Perform all initial local processing.
2. Generate candidate relation schedules by isolating the attributes first and then creating simple queries.
3. For each relation R_i,

a. Use procedure SERIAL_RR and create candidate schedules.
b. Use procedure TOTAL_RR to integrate candidate schedules.

3.1 Procedure SERIAL_RR

1. Order relations R_i such that

$$S_1 \leq S_2 \leq \ldots \leq S_m$$

where S_i is the relation to be joined with R_i

2. If no relations are at the result node, then select strategy:

$R_1 \square R_2 \square$ - - - $R_n \square$ result node

Or else if R_r is a result at the result node, then there are two strategies:

$R_1 \square R_2 \square - - - \square R_r \square - - - R_n \square R_r$

Or

$R_1 \square R_2 \ldots \square R_{r-1} \square R_{r+1} \square \ldots R_n \square R_r$

Select the one with minimum total time.

3. Build a list, L, where $L_{Ri\,Ri+1\,j}$ is set to 1 when transmission was done from R_i to R_{i+1} on join attribute *j*.

4. When calculating transmission cost,

If $L_{Ri\,Ri+1\,j} = 1$ then

cost = 0

Else

If security level of $R_i > R_{i+1}$

$cost = C_0 + C_1 * b_{ik} + (b_{ik} * ?_{(i+1)\,k})/8$

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.

$(b_{ik} * ?_{(i+1)\,k})/8$ 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 on R_{i+1} and transferred to R_i only when another join is needed between those two relations.

5. Select the strategy with minimum total time.

3.2 Procedure TOTAL_RR

1. Add candidate schedules: For each relation and candidate schedule, if the schedule contains a transmission of a joining attribute of the relation then add another similar schedule without the transmission of a joining attribute of the relation.

2. Calculate the cost of the newly added schedules as shown in step 4 of procedure SERIAL_RR.

3. Select the best candidate schedule that minimizes the total time for each joining attribute.

4. Update the list L: Set to 1 the values corresponding to all transmissions of the $BEST_{ij}$ selected.

5. Candidate Schedule Ordering: For each relation R_i, order the candidate schedules $BEST_{ij}$ on joining attribute d_{ij} so that,

$ART_{i1}+C\ (s_1*SLT_{i1}) \leq \ldots \leq ART_{is}+C\ (s_i*SLT_{is})$

where:

ART is the arrival time of the best schedule.

SLT is the accumulated attribute selectivity of the best schedule.

s_i is the selectivity of the corresponding relation.

6. Schedule Integration: For each $BEST_{ij}$ construct an integrated schedule to R_i that consists of parallel transmission of candidate schedule $BEST_{ij}$ and all schedules $BEST_{ik}$ where $k < j$.

As it can be seen, the RR algorithm does not eliminate redundant transmissions from the schedules but it makes their cost 0 when they occur. This can be made possible by adding a little overhead on the transmission cost, which is the backward cost. Using this fact, if a transmission was done from site *A* to site *B* using a join attribute *j*, then every other

transmission from *A* to *B* using *j* will have a zero cost and every transmission from *B* to *A* using *j* will have also a zero cost (if *A* and *B* have the same security level). From this point, the join can be seen as a non-redundant symmetric function. This fundamental property allowed us to enhance over traditional methods.

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.

3.3 Complexity Analysis of the RR Algorithm

As far as complexity is concerned, we can say that, without loss of generality, algorithm RR takes no more than O ($s m^2$) where:

s: number of different simple queries.

m: number of relations in the query.

4 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 contains 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.

Also consider the following query: List the product number, 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:

a- cost added per byte transmitted.

b- 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	0.4	100	0.2
R2	140	2000	400	0.4	450	0.9
R3	150	3000	900	0.9	-	-

Fig. 1. Relations Description

For each relation we have as given:

|Ri|: cardinality of the relation (number of tuples).

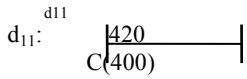
Si : size of the relation in bytes.

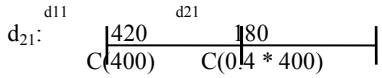
bii : for each joining attribute, the size, in bytes, of the column in the corresponding relation.

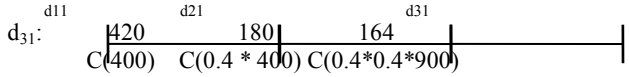
ρii : for each joining attribute, the corresponding selectivity.

Applying RR to this query, two simple queries are formed for attributes d_{i1} and d_{i2}. In step 2 of the algorithm, the following serial candidate schedules are formed:

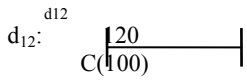
For d_{i1},

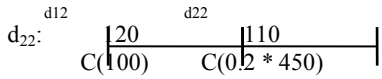






For $d_{i2,}$

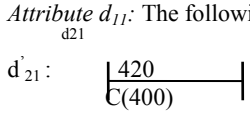


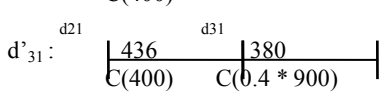


Next, we will start the construction of the schedules for each relation. Before proceeding, list L is initialized to 0 for all its entries.

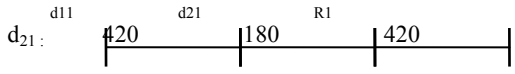
For Relation R_1:

Attribute d_{11}: The following schedules are added:

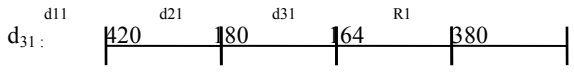




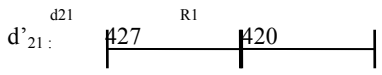
Each of the schedules of d_{11} is applied to R_1.



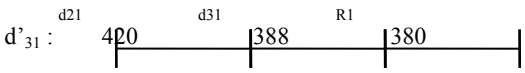
Total time = C(400)+C(0.4*1000)+C(0.4 *2000)
= 420 + 180 + 420
= 1020



Total time = C(400)+C(0.4*400)+C(0.4*0.4*900)+C(0.4*0.9*1000)
= 420 + 180 + 164 + 380
= 1144



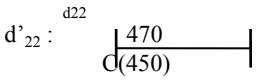
Total time = C(400 + 140 * 0.4/8) + C(0.4 * 1000)
= 427 + 420
= 847



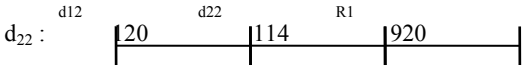
Total time = C(400)+C(0.4*900+150*0.4/8)+ C(0.4*0.9*1000)
= 420 + 388 + 380
= 1188

Choosing the minimum time schedule, we find that $BEST_{11}$ is d'_{21} with time 847.

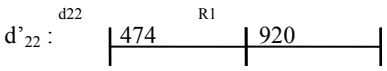
Attribute d_{12}: The following candidate schedule added:



Each of the schedules of d_{12} is applied to R_1.



Total time = C(100)+ (0.2*450+140*0.2/8)+C(0.9*1000)
= 120 + 114 + 920
= 1156

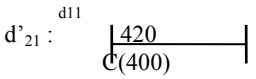


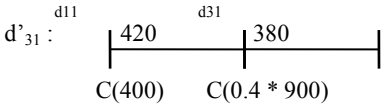
Total time = C(450 + 140 * 0.2/8) + C(0.9 * 1000)
= 474 + 920
= 1394

We find $BEST_{12}$ is d_{22} with time 1156.

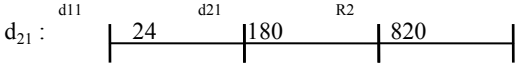
Finally, for R_1, we choose $BEST_{11}$ with time 847.
At this stage we update $L_{2-1,\,1} = 1$ and $L_{1-2\,,\,1} = 1$.

For Relation R2:
Attribute d_{11}:



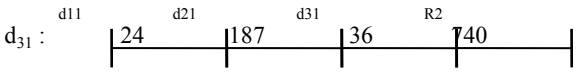


Each of the schedules of d_{11} is applied to R_2.

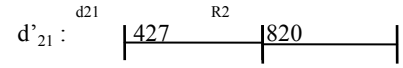


Total time=C(70*0.4/8)+C(0.4*400)+C(0.4* 2000)
= 24 +180 + 820
= 1024

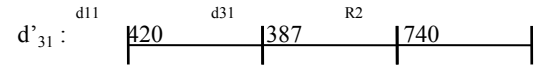
Note that transmission from 1–2 on attribute 1 is 24 which is the transmission cost of the bit vector stored in site 2 because $L_{1-2,\,1} = 1$.



Total time = C(70*0.4/8)+C(0.4*400+140*0.4*0.9/8) + C(140*0.9/8) +C(0.4*0.9 * 2000)
= 24 + 187 + 36+ 740
= 987



Total time = C(400 + 140 * 0.4/8) + C(0.4 * 2000)
= 427 + 820
= 1247



Total time =C(400)+C(0.4*900+150* 0.4*0.9/8)+C(0.4*0.9*2000)
= 420 + 387 + 740
= 1547

We find $BEST_{21}$ is d_{31} with total time 987. Also, we find $BEST_{22}$ d'_{22} with total time 1948. Finally, for R_2 we choose d_{31} with total time 987.
$L_{2-3,\,1} = 1$ and $L_{3-2,\,1} = 1$

Applying for relation R_3 we get $BEST_{31}$ with total time 740.

Hence, the most optimal total time with RR algorithm for this query is:
847 + 987 + 740 = 2574.

Using the unoptimized method we would get: 6060. Therefore, our contribution is: (6060 – 2574) / 6060 = 57.5% 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.

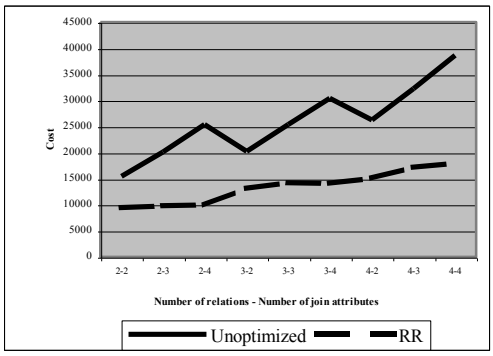
5 Experimental Results

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++ 4.0 under Windows 95. Experiments were conducted on a Pentium V PC with 64 MB RAM.

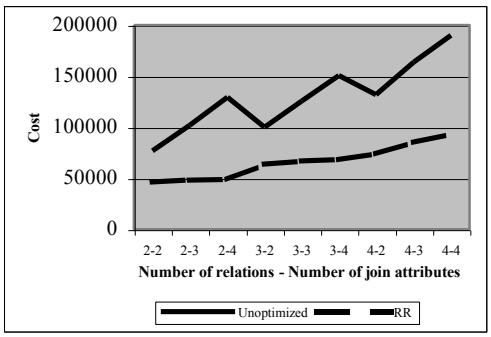
5.1 Scenario 1

In this scenario the attribute width is taken as 1 byte for all attributes. Graphically, the results are represented as follows:



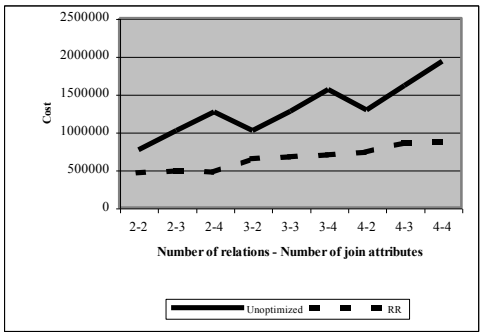
5.2 Scenario 2

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



5.3 Scenario 3

In this scenario the attribute width is taken as 50 bytes for all attributes. Graphically, the results are represented as follows:



We used three different scenarios in order to study the performance of the algorithms from different perspectives. For each scenario, we compared the performance of the algorithms with respect to the unoptimized solution. Using different scenarios we studied

better the behavior of all algorithms under a variety of circumstances. We could be able to note that RR has the best performance for a field width of 50 bytes. This result was expected because of the overhead added to the backward phase. Remember that RR provides for returning back to the original site a bit vector representing the matching tuples. This overhead is somehow more considerable when the original field width is relatively small is size because it might be more profitable sometimes not to send back this data. But when having a width of 50 bytes, the backward cost becomes negligible as compared to the forward cost.

6 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 a considerable enhancement over the unoptimized methods. Different series of experiments were conducted, allowing us to study even better the efficiency of the algorithm from different perspectives and to consider the best case for which it performs at best. We could then, based on our experiments recommend the use of the RR algorithm for huge textual and graphic distributed secured databases where the width of some join attributes is quite large, as well as for ordinary data.

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 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. Apers, P., Hevner. A., Bing Y.: Optimization Algorithms for Distributed Queries. IEEE Transactions on Software Engineering, Vol. Se-9, No. 1 (1983)
2. Bell, D., LaPadula, L.: Secure Computer Systems: Unified Exposition and Multics Interpretation. The Mitre Corporation (1976)
3. Haraty, R.: Concurrency Control and Query Processing in Multilevel Secure Kernelized Databases. Proceedings of the Symposium on Applied Computing. Phoenix, AZ (1994)
4. Hevner, A., Wu, O., Bing Y.: Query Optimization on Local Area Networks. ACM Transactions on Office Information Vol. 3, No. 1 (1985)
5. Li, Z., Ross, K.: 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)
7. Perrizo, W., Panda, P.: Query Acceleration in Multilevel Secure Database Systems. Proceedings of the 16th National Computer Security Conference. Maryland (1993)