2Q-1

Processing Distributed Set Queries -The Set Equality Correlated Case-

Mohamed El-Sharkawi Yahiko Kambayashi
Faculty of Engineering, Kyushu University

1- Introduction

To build an efficient distributed system, problems associated with distributed databases have to be solved. Among these problems, the problem of efficient query processing is important. In this paper, we discuss efficient processing of set queries. Since processing set queries involves transmition of large volumes of data, finding efficient ways to process these queries is important. We discuss set queries in context of SQL, the approach, however, is general. A set query in SQL is represented by the nested form; i.e. as two query blocks connected by a set operation. The objective is to minimize transmission cost by minimizing size of data to be transmitted between sites in the system.The approach is similar to the one in case of centeralized set queries [1]. Functional dependencies between attributes in the query are used to minimize size of transmitted data. A set query is transformed into either a distributed non-set query or a local set query. Optimization of distributed nested SQL queries was discussed in [2], however, optimization of set SQL queries is missing. Optimization of set queries, in general, was discussed in [3]. The objective is to find a transmission plan that minimizes the communication cost. It is assumed that relations in the database are partitioned.

Set queries are classified into two main types, correlated and uncorrelated. A query is correlated when the inner query block refers to a relation in the outer block, and uncorrelated otherwise. Here, correlated set queries are considered. Correlated queries are then classified into two types, one when the WHERE clause of the outer block contains a constant, the second when it contains an attribute. These types are called type-SCOR1 and type-SCOR2, respectively. The general form of these two types are:

Type-SCOR1
```
SELECT RI.C  FROM   RI
WHERE "Constant" SET OPERATION
            SELECT  RJ.K  FROM    RJ
            WHERE   RI.V = RJ.U
```
Type-SCOR2
```
SELECT RI.C  FROM   RI
WHERE RI.K  SET OPERATION
          SELECT RJ.K  FROM    RJ
          WHERE RI.V = RJ.U
```
Here, we concentrate on queries where the set operation is =ALL (i.e. set equality operation).

1-2 Assumptions

In this paper we make the following assumptions:
(1) The underlying communication network is a long-haul network. (2) The set query is of level one nesting. That is, it consists of two nested blocks. Each relation (set of relations) in a block are stored in a different site from the other block. (3) All local and non-set distributed operations are done before optimizing the set operation. (4) The cost of processing the query at the output site is always less than the cost of sending the output relation to the other site and then send the output back.

1-3 Basic Idea

The very intuitive approach to process a set query is to send either relation to the other site. If the output is needed at site RI, it is better to send RJ to site RI. If, however, the cost of sending RJ is greater than the cost of sending RI to RJ in addition to the cost of sending the output back to site RI, it is better to send RI to site RJ. In this approach, the whole relation, either RI or RJ, is sent to the other site. Our approach is to use set semantics to minimize size of one of the relations before it is sent. For two sets to be equal, the necessary condition is that they should have same size. In the relational model, the size of a set of values of a certain attribute that is associated with a value of another attribute can be reasoned using functional dependencies between the attributes. For instance, if there is an FD between attributes U and K, i.e. U→K, for any value of U there is one and only one associated value of K. On the other hand, if such dependency is not satisfied, the set of K-values associated with a u-value may be equal or greater than one. We use this knowledge to send only sets that have sizes match the set operation. There are four possible combinations of set sizes. Either the both sets are of size one, one is of size one and the other is of size greater or equal one, or the both sets are of size greater than one. For each case procedures that minimize communication cost are presented.

2- Processing Queries of The First Type

The query has the form:
```
SELECT RI FROM    RI
WHERE "Constant" = ALL
                    SELECT RJ.K  FROM    RJ
                    WHERE  RI.V= RJ.U
```
In this case Set 1 is of size one and the only element in this set has its value specified to be "Constant". The following procedure converts the query into a non-set query:
(1) Send the value "Constant" to site RJ.
(2) If RJ.U → RJ.K, apply the following query to generate a new relation RJ*(U):
```
     SELECT    RJ.U  FROM       RJ
     WHERE     RJ.K = "Constant".
```
(3) Else, apply the following query:
```
     SELECT        RJ.U  FROM    RJ
     GROUP BY      U
     HAVING        COUNT (DISTINCT K) = 1
     AND           K = "Constant"
```
In this query, DISTINCT clause is important since we handle sets. The query gets all U values having the associated sets of K values are of size one, and the only element in the set is equal to "Constant".
(4) Send the result of the query to site RI, the query becomes a local nested non-set query:
```
SELECT RI.C   FROM    RI
WHERE RI.V =    SELECT RJ*.U  FROM  RJ*
```
Cost of the procedure is the cost of sending the"Constant" from RI to RJ in addition to the cost of sending the result of the query generating RJ*.

3- Processing Queries of The Second Type

A query of this type has the form:

```
SELECT RI.C   FROM   RI
WHERE  RI.K = ALL SELECT RJ.K
                  FROM   RJ
                  WHERE  RI.V = RJ.U
```

Size of Set 1 in this case is not specified, so we need to consider all possible combinations of set sizes.

[1] Both sets are of size equal to one. This arises when the following two dependencies are satisfied: RI.V $\rightarrow$ RI.K and RJ.U $\rightarrow$ RJ.K. In this case, = ALL can be replaced by the scalar equality, and the query becomes a non-set correlated query. The cost of this conversion is cost of sending the informative messages between sites.

[2] Size of Set 1 is one and the size of Set 2 is equal to or greater than one. The following dependencies should be satisfied: RI.V $\rightarrow$ RI.K and RJ.U $\not\rightarrow$ RJ.K. Thus, U-values that may participate in the query are those having the associated set of K-values is of size one. The following procedure converts the query into a non-set query:

(1) Send a message from site RI to site RJ informing the existence of the FD RI.V $\rightarrow$ RI.K.

(2) Process the following query on relation RJ to generate a new relation RJ*(U,K):

```
RJ*(U,K) = SELECT RJ.U , RJ.K FROM   RJ
           GROUP BY U
           HAVING COUNT (DISTINCT K) = 1
```

RJ*(U,K) is the projection of RJ on U and K such that each u value has only one associated k value.

(3) Send RJ* to site RI

(4) The query becomes a centeralized non-set correlated query as follows:

```
SELECT RI.C   FROM   RI
WHERE  RI.K =   SELECT RJ*.K  FROM  RJ*
                WHERE  RI.V = RJ*.U
```

The cost of the procedure is the summation of the cost of sending the informative message from site RI to site RJ and the cost of sending RJ* to site RI.

We can minimize the cost of sending RJ* if the dependency RJ.K $\not\rightarrow$ RJ.U is satisfied, by sending an unnormalized relation of RJ* by applying Group-By on attribute K. Note that, if the number of distinct K values is small with respect to the number of U values, we can apply semi-join between RJ*and RI on attribute K in order to minimize the size of RJ* before sending it to site RI.

[3] The third case, when Set 1 is of size greater or equal to one and Set 2 is of size one. That is the following dependencies are satisfied: RI.V $\not\rightarrow$ RI.K and RJ.U $\rightarrow$ RJ.K. In this case, we apply a procedure similar to the previous one on relation RI.

(1) Send a message from RJ to RI informing the existence of the FD: RJ.U $\rightarrow$ RJ.K.

(2) At site RI apply the following query:

```
RI*(V,K,C) =
       SELECT     RI.V, RI.K, RI.C    FROM    RI
       GROUP BY   V
       HAVING     COUNT(DISTINCT K)=1
```

The query gets tuples of RI with V values has only one associated k value.

(3) Send the projection of RI* on attributes V, K, named RI**, to site RJ. The projection of RI* is sent to site RJ, violating assumption 4, since the size of RI* is small with respect to the size of relation RJ.

(4) Apply the following query at RJ to get RI#(V).

```
SELECT RI**.V    FROM    RI**
WHERE  RI**.K = SELECT RJ.K  FROM    RJ
                WHERE  RI**.V=RJ.U.
```

RI# contains V values such that their tuples satisfy both conditions in the query.

(5) Send RI# to site RI.

(6) Perform the following query at RI to get the result:

```
SELECT RI**.C    FROM    RI**
WHERE  RI**.V = SELECT RI#.V FROM RI#.
```

The cost of this procedure is the cost of sending relation RI** to site RJ in addition to the cost of sending relation RI# to site RI.

[4] The forth case when the both sets are of sizes greater than one. The situation is different from the previous cases, elements in the two sets have to be compared for equality. The following procedure minimizes the cost of data transmission and the query is converted into a local set query.

(1) Exchange messages between the two sites to inform that the set size is greater than one.

(2) Apply the following query at site RJ:

```
RJ*(U,CK,K)=
    SELECT RJ.U, COUNT(DISTINCT K), K
    FROM   RJ
    GROUP BY  U.
```

The objective of this query is to create a new attribute that counts the size of the set of K values associated with a u value.

(3) At site RI, get a relation RI*(V,CK,C) by applying a query similar to the one in Step 2 on relation RI.

```
RI*(V,CK,K,C) =
SELECT RI.V, COUNT(DISTINCT K), RI.K, RI.C
FROM   RI
GROUP BY  V.
```

This query is similar to the query in step (2).

(4) Send the projection of RJ* on attributes U, CK, named RJ#, to site RI.

(5) Apply the following query at site RI:

```
RI**(U,V,K,C) =
            SELECT  RJ#.U, RI*.V, RI*.K, RI*.C
            FROM    RI*, RJ#
            WHERE   RJ#.U = RI*.V
            AND     RJ#.CK = RI*.CK.
```

The query produces a tuple consisting of RJ#.U, RI*.V, RI*.K, and RI*.C, in which RJ#.U is equal to RI*.V and the sizes of the sets of K values associated with two similar u and v values are same.

(6) Send the projection of RI** on U, as RJ%, to site RJ.

(7) Apply the following query at site RJ:

```
RJ**(U,K) =    SELECT  RJ*.U, RJ*.K
               FROM    RJ*, RJ%
               WHERE   RJ*.U = RJ%.U.
```

The query combines each u value in the result of the query in step(5) with its associated set of K values.

(8) Send RJ** to site RI and apply the following query:

```
SELECT RI*.C FROM    RI*
WHERE  RI*.K =ALL SELECT    RJ**.K
                  FROM   RJ**
                  WHERERI*.V = RJ**.U.
```

This query obtains the final result; it is a local set query.

The cost of this procedure is the summation of transmission costs in Steps (1), (3), (6), and (8).

References

[1] El-Sharkawi, M., Kambayashi, Y., LA Symp. Feb. 1988.

[2] Gavish, B., Segev, A., ACMTODS, 11, 3, Sep. 1986.

[3] Lohman, G., M., et.al. , Proc. VLDB, 1984.