OPTIMIZATION OF JOIN STRATEGIES IN DISTRIBUTED DATABASES

PASQUALE LEGATO,1 GIUSEPPE PAILA2 and LUIGI PALPOLI†
1Dipartimento di Elettronica Informatica e Sistemistica, Università della Calabria, 87036 Rende (CS), Italy.
2Dipartimento di Informatica ed Applicazioni, Università di Salerno, 84100, Salerno, Italy.
(Received 26 July 1989; in revised form 15 October 1990)

Abstract—The paper presents a structured approach to the problem of minimizing the join cost in a relational distributed environment. A tree model is used to represent a query and a set of tree equivalence classes for query representation is identified corresponding to the space of all the feasible strategies to execute the query. The optimal strategy is then chosen by a dynamic programming approach which exploits the properties of the tree model, although the computational complexity remains exponential in the size of the problem.

Key words: Distributed databases, query optimization, dynamic programming

1. INTRODUCTION

A relational distributed database system (RDDS) is a relational database system [1] where relations are dispersed amongst a number of independent processing sites which are interconnected by a network. The interconnection network makes possible information exchanges between sites. Assuming that the distributed system consists of the sites \{S_1, \ldots, S_m\}, a query performed at site \(S_\alpha\), \(1 \leq \alpha \leq m\), represents a request for the retrieval and the processing of data, eventually located at other sites, which need to be displayed at \(S_\alpha\).

The design of an efficient strategy for processing queries has a strong impact on the performance of a distributed database system [2]. A number of algorithms to process queries in RDDS have been proposed in recent years. These algorithms can be classified along several directions: the interconnection network of reference (e.g. geographical net [3], LAN [4, 5] and star net [6]); the employed cost model (e.g. communication costs [7] vs both communication and local processing costs [8]); the objective function to be minimized (e.g. system response time [9, 10] and query processing cost [11]); and static [7, 11] vs dynamic [2] query planning. An exhaustive survey on this subject can be found in Ref. [12].

In this paper we confine ourselves to the problem of optimizing join queries only, i.e. queries implying the execution of join operations [1, 13]. The distributed optimization process for join queries can be divided into four main steps:

(a) Sequencing optimization: the best sequence of binary joins is selected to execute \(n\)-ary joins.
(b) Optimal materialization: for each relation to be retrieved, which may exist in multiple copies in the system or may be fragmented horizontally or vertically, the optimal site for materializing it is chosen.
(c) Distribution: the optimal allocation of binary join executions as well as the storing of intermediate result relations amongst the available sites is determined.
(d) Execution strategy: a binary join can be implemented by means of a number of techniques, at different costs; for instance it can be implemented as a \textit{nested loop} join or as a \textit{merge scan} join.

We assume that at each site the binary join is implemented in a predefined way, and that the cost behaviours are known. Furthermore we assume that relations are not fragmented, while allowing

†To whom all correspondence should be addressed.
the presence of multiple-copy relations: the execution model we use takes into account the problem of optimal materialization of multiple-copy relations during the distribution optimization phase. Therefore we are only focusing on points (a) and (c) above. The aim of our work is to perform an analysis of the complete space of all the feasible strategies of query execution, and to develop a search method that possibly explores a restricted number of them in order to determine the optimum. We shall propose an algorithm that finds the optimal execution strategy for distributed join queries which is static, and thus we assume that statistics and estimations of the dimension of intermediate result relations are given. The cost functions consider both local processing costs and communication costs between sites. Constraints on the storage capabilities of sites as well as constraints on the response time of the system are not considered.

The paper is organized as follows. In Section 2 we present the query execution model. In Section 3 the properties of the optimization phase are discussed and the algorithm is presented. The effectiveness of the proposed approach in real case applications is analysed in Section 4. In Section 5 some conclusions are given. The study on the cardinality of the solution space is reported in the Appendix.

2. SOLUTIONS

Before analysing the solution space, we have to point out that we shall treat joins as a pure binary operator, i.e. we shall not take care of the indexes which normally appear in join arguments. Therefore the joins that we consider are commutative and associative. Note that we do not lose any generality since indexed joins can be made commutative and associative by opportune reindexing and relation columns reordering.

Example. Consider the relations $R_1$, $R_2$, $R_3$ of arity two. Then we have:

$$[(R_1 \bowtie_{t_2 = 2} R_2) \bowtie_{t_3 = 2} R_3] = (R_1 \bowtie_{t_2 = 2} (R_2 \bowtie_{t_1 = 1} R_3)).$$

Several definitions are next introduced, which shall be used in modelling the space of feasible solutions. First we give the representation of join queries.

Definition 1. A query tree (QT) is a finite tree in which:

(i) leaves represent the relations to be joined;
(ii) each leaf has associated the list of sites where its relation is available for materialization;
(iii) internal nodes as well as the root represent join operators; and
(iv) the root has associated a label representing the query source site, i.e. the site where the result relation has to be stored and displayed.

In a QT each internal node has two or more sons. Since joins are assumed associative and commutative, a QT can be reduced to a one-level-tree—the root representing an n-ary join that can be executed using any suitable sequence of binary joins.

Definition 2. A binary query tree (BQT) is a QT where non-leaf nodes have exactly two sons.

A BQT specifies the set of relations to be joined, the query source site, and besides, by symmetrically visiting it, the order of execution of joins—we refer to the canonical symmetric visit, i.e. left_{son}:father:right_{son} for every node. It is pointed out that, at any node, the visit right_{son}:father:left_{son} yields a completely equivalent sequencing of join evaluations (we shall call two sequencings of this kind isomorphic). The objective is to find an execution strategy for the query (i.e. to select a sequence of binary joins implementing the n-ary root join and to define the sites where the joins are to be executed and the sites where the relations are to be retrieved or stored)
whose cost is minimum. A feasible solution is represented by a \( BQT \) enriched with a number of labels which specify the join execution sites and intermediate result storing sites.

**Definition 3.** An adorned binary query tree (\( AQT \)) is a \( BQT \) in which:

(i) in addition to the source site label, a second label is associated to the root, representing the site where the root join is executed;

(ii) a couple of labels is associated to each leaf representing, respectively, the site from which the relation is retrieved and the site where it is materialized (the former site belongs to the list of sites associated to the leaf); and

(iii) two labels are associated to each internal node, representing, respectively, the site where the join is executed, and the site where the intermediate result relation is stored.

**Definition 4.** An \( AQT \) is equivalent to a \( QT \) \( Q \) (\( A \) is an \( AQT \) for \( Q \)) if they have the same query source site label, they yield the same result relation, and the same set of base relations and list of sites are associated to their leaves. Two \( AQT \), \( A_1 \) and \( A_2 \), are equivalent if they are equivalent to the same \( QT \).

Given a \( QT \) \( Q \), each \( AQT \) for \( Q \) possibly specifies a different materialization of base relations to be processed as they may differ in the site labels associated to their leaves. Thus the model takes into account the eventual presence of multiple-copy relations in the system. Let \( Q \) be a given \( QT \), then there are many \( AQT \) for \( Q \). Moreover it may happen that two equivalent \( AQT \)s are also isomorphic, i.e. they specify the same distribution of executions and storings and two isomorphic sequencings of join evaluations. Obviously in order to investigate all possible equivalent \( AQT \)s, it is convenient to partition equivalent \( AQT \)s into classes of isomorphic elements, and to consider only one representative for each class.

In the Appendix the computation of the cardinality of the set of (non-isomorphic) feasible solutions is reported. We next show the fundamental result only.

**Proposition 1.** The cardinality of the solution space for the join query optimization problem, i.e. the total number of non-isomorphic \( AQT \)s for a \( QT \) having \( n \) leaves on a distributed system with \( m \) sites is:

\[
m^{(3n-3)NTS_n} \prod_{1 \leq i \leq n} p_i,
\]

where \( NTS_n \) is an exponential succession in \( n \), developed in the Appendix, and \( p_i \) is the number of sites in the list associated to the leaf of the relation \( R_i \).

3. **OPTIMIZATION**

An intrinsic feature of the optimization problem we are dealing with is that we need the information associated to the final step in order to determine the optimal decisions associated to the intermediate steps. In fact the optimal distribution among available sites can be determined only after we know which is the processing site where the final join operation has to be executed. Furthermore the problem of the optimal distribution has to be considered for all the possible execution sequencings. Thus we immediately recognize the impracticability of considering all the possible sequencings and then to consider the problem of the optimal distribution with reference to any sequencing. Fortunately there exists a dominance property, described later in the paper, which provides the guidelines to perform the generation of the optimization strategy by means of a dynamic programming approach. The method we propose builds first the optimal strategy for the subproblems of size two, i.e. for each \( i, j, 1 \leq i, j \leq n, i \neq j \), and for each site \( \alpha, 1 \leq \alpha \leq m \), the optimal \( AQT \) for the \( QT \) having the relations \( R_i \) and \( R_j \) as the leaves, and \( \alpha \) as the root label, is generated. Then the subproblems of size three are solved and so on. In order to solve the subproblems of any size \( k \), the dominance property allows us to use the already generated solutions for the subproblems of size smaller than \( k \). In the next section we shall show how the optimal
distribution for a given sequencing can be generated using the cost model. In Section 3.2 we shall introduce the dominance property and shall show how it can be used to build the optimal AQT for the whole given QT.

3.1. Optimal distribution

We assume that the cost of a join operation performed in a distributed environment is given by

(i) local processing cost and
(ii) transmission cost,

and the two above costs are additive. All the costs depend on the dimensions of the relations they refer to, but we do not assume any particular form of dependency (e.g. linear) of costs from dimensions. Finally, since we have assumed a heterogeneous distribution environment, the processing cost depend upon the site where the join operation is performed. Furthermore the transmission cost of an operand relation between a couple of sites also depends upon the specific couple.

Remark. The amount of data needed in our cost model is quite large; the reason is that we are considering the most general structure of the distributed system, and the required data are precisely what is needed in this framework. If some restriction is imposed on the structure system, this amount decreases consequently. For instance, in the presence of a high-speed LAN distributed system, the communication cost is no longer an important factor to be taken into account when computing the execution cost of a distributed join operation. As a consequence, the amount of data needed decreases sensitively. However, we want to stress that the optimization method, presented later on in this paper, is completely parametrical w.r.t. the structure of the cost model. In fact, the only cost information accessed by our algorithm is “the cost to obtain a certain (possibly intermediate) relation stored at a particular site”, without any reference to the way this cost is to be specifically computed.

Given a particular sequencing of join execution (see Fig. 1), and using the cost model, the join execution site \( x_{F_j} \) and the partial result relation storing site \( x_{S_j} \) for the join associated with each internal node \( i \), the execution site \( x_{P_{root}} \) associated to the root join, and the couple of sites for base relation materializations \((x_{S,l}, x_{M,l})\) associated to each leaf \( L \) must be chosen, in such a way that the whole cost associated with the root of the resulting AQT is minimum.

Using the dominance property presented in the next section, it is possible to reduce the whole problem of finding the optimal distribution for a given sequencing, to the solution of a number of subproblems and in particular we have only to find the optimal distribution for subsequencings of size two. The optimal distributions of size two are then met in several ways in order to obtain the optimal distributions for the sequencing of size \( n \).

The start step of the procedure consists of computing for each site \( \alpha, 1 \leq \alpha \leq m \), the optimal materialization of each base relation \( R_i, 1 \leq i \leq n \), w.r.t. \( \alpha \). Let \( TC(\beta \rightarrow \gamma) \) be the minimum cost paid to move the relation \( R_i \) through the possible paths from site \( \beta \) to site \( \gamma \). Assume that the list of sites associated to the leaf of \( R_i \) is \((x_{\alpha_1}, \ldots, x_{\alpha_k})\) then the optimal materialization is the one associated to the cost \( C_i^\gamma \) defined as follows:

\[
C_i^\gamma = \min_{1 \leq r \leq k} TC(x_{\alpha_r} \rightarrow x_i).
\]

The core step is illustrated next. Consider the tree portion in Fig. 2, drawn from any given sequencing of the type shown in Fig. 1, and assume that we have already computed the minimum cost to obtain the (stored) partial result relations \( R_i, R_j \), at each of the available sites \((1, 2, \ldots, m)\).

We stress that \( R_i \) and \( R_j \) may be either base relations or intermediate result relations. Assume that we want to compute the optimal distribution for a QT involving \( R_i \) and \( R_j \) with site \( 1 \) as the query source site. The computation of the optimum cost associated with the root of the tree proceeds as follows (w.l.o.g. we refer to a case where \( m = 2 \)). Let \( PC_i^\gamma \) be the cost paid to execute a join
between \( R_i \) and \( R_j \) at site \( x \). The cost \( C_{ij}^p \) to obtain the relation \( R_{ij} \) stored at site 1 has to be computed. To evaluate \( C_{ij} \), we need to compare two groups of alternatives:

Group \( A \), the join processing site is 1,

\[
\begin{align*}
\text{cost}[A_1] &= C_i^j + C_j^i + PC_{ij}^p \\
\text{cost}[A_2] &= C_i^j + C_j^i + PC_{ij}^p + TC_j(2 \rightarrow 1) \\
\text{cost}[A_3] &= C_i^j + C_j^i + PC_{ij}^p + TC_i(2 \rightarrow 1) \\
\text{cost}[A_4] &= C_i^j + C_j^i + PC_{ij}^p + TC_i(2 \rightarrow 1) + TC_j(2 \rightarrow 1);
\end{align*}
\]

and

Group \( B \), the join processing site is 2,

\[
\begin{align*}
\text{cost}[B_1] &= C_i^j + C_j^i + TC_i(1 \rightarrow 2) + TC_j(1 \rightarrow 2) + PC_{ij}^p + TC_{ij}(2 \rightarrow 1) \\
\text{cost}[B_2] &= C_i^j + C_j^i + TC_i(1 \rightarrow 2) + PC_{ij}^p + TC_{ij}(2 \rightarrow 1) \\
\text{cost}[B_3] &= C_i^j + C_j^i + TC_i(1 \rightarrow 2) + PC_{ij}^p + TC_{ij}(2 \rightarrow 1) \\
\text{cost}[B_4] &= C_i^j + C_j^i + PC_{ij}^p + TC_{ij}(2 \rightarrow 1);
\end{align*}
\]

where \( C_{ij}^p \) is the optimum cost to have \( R_i \) stored at \( \beta \), \( z = i, j \), \( \beta = 1, 2 \).

Note that \( \text{cost}[A_1] < \text{cost}[A_4] \) and \( \text{cost}[B_4] < \text{cost}[B_1] \), \( 2 \leq k \leq 4, 1 \leq h \leq 3 \), since we are assuming that \( C_i^j \) and \( C_j^i \) are optima, \( \beta = 1, 2 \). Therefore, in the general case, only \( m \) alternative patterns have to be considered at each step. Given the best out of the \( A \) and \( B \) alternatives, by comparison we obtain the best alternative with respect to site 1.

Therefore, given a problem of dimension two, we are able to solve it. In the next section we shall show how, using the dominance property, it is possible to solve problems of any size \( k \), partitioning them in subproblems of smaller size.

---

**Fig. 1.** Elements of \( TS_k \), \( 0 \leq k \leq 5 \), and \( lrs \) for \( \{R_1, R_2, R_3, R_4\} \).

**Fig. 2.** Partial tree portion.

**Fig. 3.** An AQT.

**Fig. 4.** An AQT.
3.2. The dominance property

In this section we show how the optimal AQTV for a given QT can be generated. To this end a dominance property is presented allowing an incremental construction of an optimal AQTV.

**Proposition 2** (dominance property). Assume that an optimal AQTV OT for a given QT with source site α is given, and consider a sub-AQTV ST rooted at i; if α_{yj} = β in OT, then ST is an optimal AQTV for a query performed at β, and involving the subset of relation spanned by ST.

**Proof.** We proceed by contradiction, and assume that ST is not optimal. Then another AQTV BT shall exist, which involves the same subset of relations, but less cost. Thus by substituting in OT the tree BT in place of ST, a new AQTV is obtained which is less costly than the optimal one (contradiction). □

In order to clarify the role of the dominance property in designing an optimization procedure consider a query involving the set \{R_1, R_2, R_3, R_4\} to be joined in a system restricted to have only one site (thus the site labels are not reported). Assume that the optimal AQTV is the one reported in Fig. 3. Consider now the sub-AQTV in Fig. 4. The dominance property assures that this latter sub-AQTV would be an optimal AQTV for a query involving the set \{R_1, R_2, R_3\}. Vice versa assume that the optimal tree for a query involving \{R_1, R_2, R_3, R_4\} has to be determined, but we know in advance that the left and right subtrees of the optimal tree span the subsets \{R_1, R_2, R_3\} and \{R_4\}, respectively. If the optimal tree for \{R_1, R_2, R_3\} has already been determined and it is equal to the one reported in Fig. 4, then the dominance property implies that the optimal AQTV spanning \{R_1, R_2, R_3, R_4\} is the one reported in Fig. 3.

Unfortunately there is no way to know in advance what the binary partition of the initial set of relations is that would give the optimal AQTV for a given QT. Therefore the method we propose generates an optimal AQTV for QT by considering all the possible binary partitions of the initial set of relations.

Moreover the assumption that the optimal tree for \{R_1, R_2, R_3\} has been already determined suggests a bottom-up construction of sub-AQTVs spanning larger and larger subsets of relations. Say k is the dimension of the currently considered relation subsets: there are \(\binom{k}{l}\) different subsets with this cardinality. If the distributed system at hand consists of m sites, then we have to determine \(m(\binom{k}{l})\) optimal sub-AQTVs. Once fixed just one out of the above subsets of relations with k elements, say \(R_1^k, 1 \leq l \leq \binom{k}{l}\), we generate all the binary partitions of this subset. The number of these partitions is

\[
\sum_{1 \leq l \leq \binom{k}{l}} \binom{k}{l}.
\]

Note that for each partition \((R_1^{k-l}, R_l^k)\) it turns out that \(R_1^{k-l}\) and \(R_l^k\) span a number of relations \(<k\), therefore the optimal sub-AQTVs for these subsets have been already computed, and thus the dominance property applies as shown above—note that we consider only non-trivial partitions, i.e. partitions \((R_1^{k-l}, R_l^k)\) in which neither \(R_1^{k-l}\) nor \(R_l^k\) is empty.

3.3. The algorithm

The method described above is summarized in the following procedure:

**Begin**
Let \(R\) be the set of \(n\) relations to be joined. Let \(\alpha_{\text{root}}\) be the query source site.

(1) for \(i = 1, \ldots, n\)
   - Compute the best materialization for \(R_i\), \(1 \leq i \leq n\), w.r.t. each site \(\alpha\), \(1 \leq \alpha \leq m\), and the cost \(C_i^\alpha\) associated.

(2) for \(k = 2, \ldots, n\)
   - for \(l = 1, \ldots, \binom{k}{l}\)
     - Set \(\text{current}\_\text{solution}_i^{\alpha} = \text{None}\), Set \(\text{current}\_\text{cost}_i^{\alpha} = \infty\), \(1 \leq \alpha \leq m\);
     - Compute, w.r.t. each site \(\eta\), \(1 \leq \eta \leq m\), if \(\alpha \neq \eta\), w.r.t. the only site \(\alpha_{\text{root}}\), if \(\alpha = n\), the best sub-AQTV associated with the subset \(R_i^k \subseteq R\), with \(k\) elements as follows:
for each couple of non-empty subsets \( R_i \) and \( R_j \) which constitute a partition of \( R_i \) do:

(i) Consider the optimal sub-AQT spanning \( R_i \) and the optimal sub-ABT spanning \( R_{i+\delta} \) w.r.t. site \( \delta, 1 \leq \delta \leq m \);

(ii) Compute the cost of this solution and the distribution associated as shown in Section 3.1;

(iii) Compare this cost with \( \text{current}_{\text{cost}}^{i,x} \). If it is better then update \( \text{current}_{\text{cost}}^{i,x} \) and \( \text{current}_{\text{solution}}^{i,x} \).

(3) Output (\( \text{current}_{\text{solution}}^{i,x} \)).
End.

The time complexity analysis is given below, considering the operation of analysing a single cost pattern (e.g. \( A_1 \rightarrow A_4, B_1 \rightarrow B_4 \) above) as having a unit weight. As far as the space complexity is concerned, we assume that the storing of one sub-AQT with \( k \) leaves has weight \( k \).

**Proposition 3.** The time complexity of the algorithm is

\[
O \left( m^2 \sum_{2 \leq k \leq n-1} \binom{n}{k} \sum_{2 \leq i \leq \lfloor k/2 \rfloor} \binom{i}{k/2} \right).
\]

**Proof** (step 1). For each materialization site \( a, 1 \leq a \leq m \), and for each relation \( R_i, 1 \leq i \leq n \), we have to compare \( p_i \) cost patterns, one for each site where \( R_i \) can be retrieved, then the procedure considers a total of \( m \sum_{1 \leq i \leq n} p_i \) cost patterns.

(2) Disregarding execution and storing sites, the inner computation generates \( \sum_{2 \leq i \leq \lfloor k/2 \rfloor} \binom{i}{k/2} \) sub-AQTs if \( i \neq k/2 \) (when \( i = k/2 \) a 1/2 factor should be multiplied to avoid the duplication of operations). When sites are considered by the reasoning developed in the presentation of the cost model, we have to consider \( m \) in triplets (left operand site, right operand site; execution site) of the type \((a, a, a), 1 \leq a \leq m\). Therefore we have to consider a set of \( m \) alternatives. For each of these alternatives, we have to consider \( m \) intermediate result storing sites, and one root result storing site, and then a total of \( m^2 \) alternatives for distributions associated to intermediate nodes, and a total of \( m \) alternatives for the root join. Furthermore we have to consider the number of possible subsets of relation with cardinality \( k \) chosen (second level for in the procedure). There are \( \binom{n}{k} \) subsets of this kind. Then we have:

\[
m^2 \sum_{2 \leq i \leq \lfloor k/2 \rfloor} \binom{i}{k/2} + m \sum_{2 \leq i \leq \lfloor k/2 \rfloor} \binom{i}{k/2}.
\]

Finally, the outermost for on \( k \) corresponds to a leftmost summation \( \sum_{2 \leq k \leq n} \). Therefore the overall expression is

\[
m \sum_{1 \leq i \leq n} p_i + m^2 \sum_{2 \leq k \leq n-1} \binom{n}{k} \sum_{2 \leq i \leq \lfloor k/2 \rfloor} \binom{i}{k/2} + m \sum_{2 \leq i \leq \lfloor k/2 \rfloor} \binom{i}{k/2},
\]

which reduces to

\[
O \left( m^2 \sum_{2 \leq k \leq n-1} \binom{n}{k} \sum_{2 \leq i \leq \lfloor k/2 \rfloor} \binom{i}{k/2} \right). \quad \square
\]

We notice that an upper bound for the time complexity is \( O(m^2 2^n) \).

**Proposition 4.** The space complexity of the algorithm is

\[
O \left( m \sum_{1 \leq k \leq n-1} k \binom{n}{k} \right).
\]

**Proof.** For each subset of cardinality \( k, 1 \leq k \leq n-1, m \) optimal AQTs are to be stored, one for each site. An AQT spanning \( k \) optimal AQTs has weight \( k \). There are \( \binom{n}{k} \) subsets of cardinality \( k \) w.r.t. a set of cardinality \( n \). Moreover only one optimal AQT spanning \( n \) relations is to be stored—this
$AQT$ actually represents the optimal solution for the problem at hand. Therefore the space needed is

$$
\left[ m \sum_{1 \leq k \leq n-1} k \binom{n}{k} \right] + n.
$$

Hence the result follows.

We notice that an upper bound for the space complexity is $O(mn2^n)$.

Both the time and space complexity of the algorithm are exponential in the number of relations to be joined. However, it is worth noting that:

(i) Dynamic programming yields a time complexity which is a reasonable slow exponential function. Using a "brute force" enumeration algorithm would have needed a polynomial space, but a fast exponential time complexity (at least, the factorial of the number of relations to be joined).

(ii) If we fix an upper bound to the number of relations to be joined, the time and space complexity turn out to be polynomial. This consideration may be actually applied to a real distributed query—indeed it is unusual to have a real, say, SQL query involving more than 10-15 relations.

4. DISCUSSION AND COMPARISONS

Since the time and the space complexity of the optimization algorithm presented in the previous sections are both exponential, a discussion is needed in order to assess the effectiveness of our approach in real case applications. To this end we compare our algorithm against the $R^*$ [11] optimizer, a well-established RDDS. First, we notice that the exponential factors for both the time and space complexity of our algorithm arise from the computation of the order (sequencing) according to which the binary joins are to be executed. Therefore, in developing the comparative analysis below, we avoid to consider the determination of the distribution of operations amongst the sites of the RDDS.

Now, let $\{R_1, \ldots, R_n\}$ be the set of relations to be joined. The $R^*$ optimizer finds the best way of joining larger and larger subsets of relations. The best way to join any couple of relations is determined first. Then the subsets of cardinality three are considered, and so on. The process terminates when the execution order associated to the optimal solution for the overall set $\{R_1, \ldots, R_n\}$ of base relations has been found. The method works under the assumption that at least one of the two operands of each binary join is a base relation.

On the contrary, our method allows to remove such a restriction, thus generalizing the $R^*$ approach. As a consequence, unlike $R^*$, we are always able to achieve the optimum. The price to be paid for this reduces to a quadratic exponent in the time complexity and a $n$ multiplicative factor in the space complexity [indeed, both the time and space complexity of the $R^*$ optimizer are $O(2^n)$ [11]]. Therefore the application domains of the two approaches turn out to be equivalent.

The following simple example shows the completeness of our method as opposed to the $R^*$ one.

**Example.** Let $\{R_1, R_2, R_3, R_4\}$ be a set of relations to be joined on a one-site system. Assume that the selectivity factors associated to the joins at hand, and the dimensions of the relations induce the following costs (the site label is not specified):

$$
cost(R_1 \bowtie R_2) = 1 \quad \cost(R_3 \bowtie R_4) = 1 \quad \cost((R_1 \bowtie R_2) \bowtie (R_3 \bowtie R_4)) = 2
$$

$$
cost(R_1 \bowtie R_2) = 2 \quad \cost(R_3 \bowtie R_4) = 2 \quad i \neq 1, j \neq 3
$$

$$
cost((R_i \bowtie R_j) \bowtie (R_k \bowtie R_l)) = 4 \quad (i \neq 1) \text{ or } (j \neq 2) \text{ or } (k \neq 3) \text{ or } (t \neq 4)
$$

$$
cost((R_i \bowtie R_j) \bowtie R_k) = 4 \quad \cost((R_i \bowtie (R_j \bowtie R_k)) \bowtie R_l) = 5,
$$

where parentheses indicate the join execution order. It follows that the solution determined by our approach is the one obtained joining $R_1$ with $R_3$ and $R_3$ with $R_4$ first, and then joining the two
resulting relations together. Moreover, notice that this last join operation, having two intermediate result relations as the operands, violates the assumption under which the R* optimizer works. Therefore, this (optimal) solution is not computed by R*.

5. CONCLUSIONS

The complete space of solutions has been considered in this work in order to pursue the optimization of join operations in distributed systems, under very general conditions. The problem has been formulated in terms of the construction of a tree structured solution (AQT), and a dominance property has been used to restrict the search space to be explored. An important feature of the proposed methodology lies in the possibility of using it as a starting framework to particularize the solution procedure with reference to more specific environments, and to develop heuristic procedures that are effective in the solution of large size problems. Any suitably modified strategy of construction of the tree structured solution can be a potential way to obtain a suboptimal solution. Current work is being done on this point. Fragmented relations have not been considered in this paper; nevertheless it is worth noting that the approach we presented can be extended to cope with fragmentation. Vertical fragmentation is implicitly handled in our framework. Intuitively, the problem of joining n relations in the presence of vertical fragmentation, reduces to the problem of joining $\Sigma_{1 \leq i \leq n} f_i$ non-fragmented relations, where $f_i$ is the number of vertical fragments in which the i-th relation is split. Horizontal fragmentation is a more difficult problem to be dealt with [14, 15]. Techniques similar to that presented in Ref. [16] could be used for handling horizontally fragmented relations.

Acknowledgements—The authors wish to thank Professors Stefano Ceri and Domenico Saccà for the helpful comments on the draft versions of this paper.

REFERENCES


APPENDIX

Let SE be the set of all possible sequencings, and consider the relation $\theta$ of isomorphism between sequencings. Thus $SE/\theta$ is the modulo set of $\theta$ on SE. Since to any two sequencings belonging to the same class in $SE/\theta$ corresponds the same implementation of the original n-ary join by means of binary joins (see Fig. A1), actually only one representative for each class in $SE/\theta$ must be considered. Two AQT's are isomorphic if they have consistent shapes coupled with a consistent
association between leaves and base relations, and they specify an identical distribution of materializations, executions and storings among sites. Therefore, in order to compute the number of distinct solutions, we shall proceed in several steps:

1. Computation of the number of non-consistent AQT shapes, which we shall call distinct AQT topologies.
2. Computation of the number of non-consistent associations of base relations to AQT leaves, which we shall call distinct leaf-relation assignments.
3. Computation of the total number of solutions.

Note that the computation of the cardinality of the set of AQT topologies coupled with leaf-relation assignments (points (1) and (2) above) yields the cardinality of \(SE/\theta\), i.e. the number of non-isomorphic sequencings for the QT at hand.

Non-equivalent Topologies

Before analysing the topology equivalence of AQTs, several definitions are needed.

**Definition 5.** An \(n\).leaf topology (\(nlt\)) is an unlabelled binary tree with exactly \(n\) leaves.

**Definition 6.** Two \(nlt\)s \(p\) and \(q\) are equivalent (w.r.t. sequencing) \((pq)\) either if \(p\) and \(q\) are identical or if \(p\) can be obtained from \(q\) by exchanging the left (right) subtree of one or more of its nodes with the right (left) subtree.

It is clear that the relation \(\tau\) is an equivalence relation, since it is reflexive, symmetric and transitive.

Given a QT with \(n\) leaves, the number of distinct AQT topologies is equal to the cardinality of the set \(TMS_n = NT/\tau\), where \(NT\) is the set of all \(nlt\)s. \(TS_n = \{p\mid p\in TMS_n\}\) denotes the set of the representatives of the equivalence classes in \(TMS_n\).

In order to compute the cardinality of \(TS_n\), which is clearly equal to the cardinality of \(TMS_n\), an operation between AQT topologies is next introduced.

**Definition 7.** The sum (denoted with \(\bullet\)) between a \(klt\) \(p\in TS_n\) and a \(jlt\) \(q\in TS_n\) is a \((k+j)lt\) \(p\bullet q\in TS_{n+j}\) whose root is a new node having \(p\) and \(q\) as the left and the right subtrees, respectively.

Note that, from the definition of \(TS_n\), \(n\geq 1\), it follows that \(\bullet\) is a commutative and associative operator.

**Proposition 5.** The cardinality of \(TS_n\), i.e. the number \(N_n\) of distinct \(nlt\)s is given by the following recursive formula:

\[
N_n = \sum_{r=1}^{\lfloor n/2\rfloor} \Delta^1(N_{n-r}, (N_r + \Delta^2_{n})) > 1, \quad \text{where} \quad \Delta^1 = \begin{cases} 1 & \text{if } r \neq n/2, \\ 1/2 & \text{if } r = n/2, \end{cases} \quad \Delta^2 = \begin{cases} 0 & \text{if } r \neq n/2, \\ 1 & \text{if } r = n/2. \end{cases}
\]

**Proof.** The number of distinct \(nlt\)s obtained as sum between a \((n-r)lt\) and a \(rlt\) when \(r \neq n/2\) is equal to

\[
(N_{n-r}, N_r).
\]

the total number of distinct \(nlt\)s when \(r \neq n/2\), is equal to

\[
\sum_{r=1}^{\lfloor n/2\rfloor} (N_{n-r}, N_r).
\]

Besides, when \(r = n/2\) the number of \(nlt\)s is equal to

\[
N_r + (N_r/2)(N_r - 1);
\]

in fact \(N_r\), due to commutativity, is the number of \(nlt\)s which can be obtained as the sum between two \(nlt\)s belonging to the same equivalence class in \(TMS_n\), and \(N_r(N_r - 1)/2\) is the number of \(nlt\)s which can be obtained as the sum between two \(nlt\)s belonging to different equivalence classes in \(TMS_n\).

Non-equivalent Assignments

For a given topology of an AQT, several assignments of base relations to leaves can be fixed to obtain different sequencings of the binary joins. The formal definition of leaf-relation assignment is given below.

**Definition 8.** Let \(L = \{L_1, \ldots, L_n\}\) denote the set of leaves of a given \(nlt\), numbered from left to right in the natural order. Let \(R = \{R_1, \ldots, R_n\}\) denote a set of \(n\) base relation. A leaf-relation assignment \(\rho\) is a set of couples \((L_j, R_j)\), where \(L_j \in L\), \(R_j \in R\) and for each \(L_j \in L\) (\(R_j \in R\)) there is one and only one couple in \(\rho\) with first (second) element \(L_j (R_j)\).

**Definition 9.** Given a \(p\in TS_n\) and a set of \(n\) base relations, two assignments \(\rho_1\) and \(\rho_2\) are equivalent (w.r.t. sequencing) \((\rho_1\rho_2)\) if \(\rho_1\) can be obtained from \(\rho_2\) by exchanging in a specular fashion the relations associated to leaves belonging to any two symmetric, i.e. topologically equivalent, brother subtrees in \(p\).

**Example.** Consider the 3lt \(p\) shown in Fig. A2. Then,

\[
\{(L_1, R_1), (L_2, R_3), (L_3, R_2)\}\rho_2 \{(L_1, R_1), (L_2, R_1), (L_3, R_2)\}.
\]

For a QT with \(n\) leaves and \(p\in TS_n\), the number of distinct \(\rho\)s on \(p\) is equal to the cardinality of the set \(AMS_p = N\rho/\rho\), where \(N\rho\) denotes the set of all \(\rho\)s on \(p\). \(AMS_p = \{\rho\mid \rho\in AMS_p\}\) denotes the set of the representatives of equivalence classes in \(AMS_p\).
Proposition 6. The cardinality of \( \mathcal{A}S_n^p \), i.e. the number \( T_n^p \) of distinct \( brs \) on \( p \), where \( p \in TS_n \), is obtained as the sum between \( q \in TS_{n-1} \), and \( w \in TS_n \), is equal to

\[
T_n^p = T_n^p q \cdot T_n^p w, \quad \text{where} \quad T_n^p q = 1 \quad \text{if} \ r \neq n/2 \quad \text{or} \ q \neq w
\]

\[
T_n^p w = \frac{1}{2} \quad \text{if} \ r = n/2 \quad \text{and} \ q = w.
\]

Proof. Given a set of \( n \) relations \( R \), partitioned in two disjoint subsets \( R^{n-1} \) and \( R^r \), containing, respectively, \( n - r \) and \( r \) elements, the number of assignments of \( R \) on \( p \in TS_n \), where \( p \) is obtained as sum between a \( q \in TS_{n-1} \), and a \( w \in TS_n \), and \( R^{n-1} \) is assigned on \( q \) and \( R^r \) is assigned on \( w \), is equal to

\[
T_n^p q \cdot T_n^p w.
\]

If either \( r \neq n/2 \) or \( q \neq w \), the number of distinct subsets of \( n - r \) relations among \( n \), which can be assigned on \( w \) is equal to \( \binom{n}{r} \), due to the equivalence. Therefore,

\[
T_n^p q \cdot T_n^p w = \binom{n}{r} T_n^p q \cdot T_n^p w, \quad \text{if} \ r \neq n/2 \quad \text{or} \ q \neq w;
\]

otherwise if \( r = n/2 \) and \( w = q \) the number of distinct subsets of \( r \) relations which can be assigned on \( q \) is equal to \( \binom{n}{r} \), therefore

\[
T_n^p = 1/2 \binom{n}{r} T_n^p q \cdot T_n^p w, \quad \text{if} \ r = n/2 \quad \text{and} \ q = w. \quad \square
\]

Total Number of Solutions

The total number of distinct sequencings of the \( n \)-ary join into a set of binary joins is computed next.

Proposition 7. The total number of operation execution sequencings (\( \mathcal{N}TS_n \)) on a set of \( n \) base relations to be joined, is equal to

\[
\mathcal{N}TS_n = \sum_{r=1}^{\lceil \frac{n}{2} \rceil} \left( \sum_{q \in TS_{n-1}} \sum_{w \in TS_n} A_n T_n^p q \cdot T_n^p w \right), \quad \text{where} \quad A_n = \begin{cases} 1 & \text{if} \ r \neq n/2 \\ 1/2 & \text{if} \ r = n/2. \end{cases}
\]

Proof. The number of operation sequencings for a set of \( n \) relations on the elements of \( TS_n \) is obtained as the sum between the \( N_{n-r} \) elements in \( TS_{n-1} \), with the \( N_r \) elements in \( TS_n \). Thus, by Proposition 2, we have the following terms for fixed \( r \):

\[
\binom{n}{r} \sum_{q \in TS_{n-1}} \sum_{w \in TS_n} A_n T_n^p q \cdot T_n^p w, \quad \text{if} \ r \neq n/2
\]

\[
\sum_{1 \leq r_1 < \ldots < r_1 \leq n} \sum_{N_{n-r} \in TS_{n-r}} \sum_{N_r \in TS_n} \frac{1}{N_{n-r}} T_n^p q \cdot T_n^p w, \quad \text{if} \ r = n/2 \quad \text{and} \ q \neq w.
\]

where \( f \) is an ordering injective function from \( nts \) to non-negative integers;

\[
1/2 \binom{n}{r} \sum_{w \in TS_n} T_n^p q \cdot T_n^p w, \quad \text{if} \ r = n/2 \quad \text{and} \ q = w. \quad \square
\]

Hence the proposition follows.

The elements in \( TS_n \) together with each feasible \( br \) associated to every subset of \( k \) elements of the set of relations \( \{R_1, R_2, R_3, R_4\} \), \( 1 \leq k \leq 4 \) are reported in Fig. 1. The values of \( \mathcal{N}TS_n \), \( 1 \leq n \leq 6 \) are

\[
\begin{align*}
\mathcal{N}TS_1 &= 1 \\
\mathcal{N}TS_2 &= 1 \\
\mathcal{N}TS_3 &= 3 \\
\mathcal{N}TS_4 &= 15 \\
\mathcal{N}TS_5 &= 105 \\
\mathcal{N}TS_6 &= 945.
\end{align*}
\]

We are now able to compute the total number of solutions, also taking into account the distribution of operation executions and intermediate result relations storings over sites.
Proposition 1. The cardinality of the solution space for the join query optimization problem, i.e. the total number of non-isomorphic AQTs for a QT having $n$ leaves on a distributed system with $m$ sites is:

$$m^{(3n-3)NTS} \prod_{1 \leq i \leq s} p_i$$

Proof. For each sequencing of an $n$-ary join operation, the total number of materialization sites associated to leaves is $m^{\sum_{1 \leq i \leq s} p_i}$; the total number of distributions of join executions and result relation storings associated to the $n-2$ intermediate nodes is $m^{3n-4}$; the total number of execution site for the root join is $m$. Hence the result follows. $\square$