$\pi_{S \cup Y}(T)$, then $\mu[S]$ is in $\pi_S(T)$ and $\mu[Y]$ is in $\pi_Y(T)$. Thus, μ is in

$$\pi_S(T) \times \pi_Y(T)$$

Now, we use the fact that the relations to which step (3) applies are reduced. First, we note that $\pi_S(T) = S$, since S is one of the relations in the join T, and S has no dangling tuples. Thus, the size of $\pi_S(T)$ is no greater than I, since surely S is no larger than the entire input.

Second, we note that $\pi_Y(T)$ can be no larger than U. The proof consists of two observations

1. $\pi_Y(T) = \pi_Y(R_1 \bowtie \cdots \bowtie R_k)$. The reason is that, because the relations are reduced, every tuple in T extends to at least one tuple over all of the attributes, when all of the relations $R_1, \ldots, R_k$ are joined.
2. $Y \subseteq X$, so $\pi_Y(R_1 \bowtie \cdots \bowtie R_k)$ can be no larger than $\pi_X(R_1 \bowtie \cdots \bowtie R_k)$.

But $\pi_X(R_1 \bowtie \cdots \bowtie R_k)$ is the output, so (1) and (2) imply that the size of $\pi_Y(T)$ is no greater than U. We may now invoke (11.42) to argue that $\pi_{S \cup Y}(T)$ has size no greater than $2IU$. In detail, let t_1 be the number of tuples in $\pi_S(T)$ and t_2 be the number of tuples in $\pi_Y(T)$. Let l_1 be the length of a tuple in $\pi_S(T)$ and l_2 be the length of a tuple in $\pi_Y(T)$. Then $l_1 t_1 \leq I$, and $l_2 t_2 \leq U$. Now $\pi_{S \cup Y}(T)$ has at most $t_1 t_2$ tuples, by (11.42), and their length is $l_1 + l_2$. Thus, its size is at most $(l_1 + l_2) t_1 t_2$. Finally, observe that $l_1 + l_2 \leq 2 l_1 l_2$.[39] Thus, the size of $\pi_{S \cup Y}(T)$ is no more than $2 l_1 l_2 t_1 t_2 = 2IU$. □

Theorem 11.6: For a fixed query of the form (11.38), Algorithm 11.4 requires communication cost and running time that are polynomial in the size, I, of its input relations, the size, U, of its output relations, and the number, k, of relations in the join of (11.38).

Proof: Step (1), applying the full-reducer semijoin program, requires $O(I)$ communication and $O\big(k(I \log I + U \log U)\big)$ total computation time, as we have seen. Step (2), construction of a parse tree, is implicit in the discovery of the full reducer. To find the full reducer or the parse tree takes time at most the product of the number of hyperedges times the number of nodes in the hypergraph constructed from (11.38). On the assumption that there are no empty relations in the input, the size of the hypergraph cannot exceed I, and if there are empty relations, that fact can be discovered in $O(k)$ time and the empty relation produced as the output immediately, without performing steps (3) and (4) of Algorithm 11.4. We conclude that $O(kI)$ time suffices to find the full reducer for step (1) and to find the parse tree in step (2), in all but the trivial case where there is an empty relation. Thus, we can neglect the cost of step (2), since the estimate we use for step (1) is higher, both in transmission cost [there is none in step (2)] and in computation cost.

For step (3), we must transmit $k - 1$ relations; none are larger than $2IU$, by Lemma 11.1. Thus, the total transmission cost is $O(kIU)$. This amount dominates the transmission cost of step (1), and can be taken to be, within a constant factor, the transmission cost of the entire algorithm. For the computation cost, we take $k - 1$ joins, whose input and output relations are all bounded by size $2IU$. Thus, a method such as sort-join offers running time $O\big(IU \log(IU)\big)$ per join, for a total cost of $O\big(kIU \log(IU)\big)$.

Finally, the projection of step (4) requires no communication and takes time $O(IU)$. We conclude that step (3) dominates both the transmission cost and the total computation time, so the algorithm as a whole has transmission cost $O(kIU)$ and running time $O\big(kIU \log(IU)\big)$. □

[39] Of course, usually, $l_1 + l_2$ is much less than $l_1 l_2$. However, because of the possibility that $l_1 = l_2 = 1$, we need the factor 2 so the claim holds generally.

11.15 THE SYSTEM R* OPTIMIZATION ALGORITHM

System R* is the experimental extension of System R to the distributed environment. Like its parent, it performs optimization in an exhaustive way, confident that the cost of considering many strategies will pay off in lower execution costs for queries.

The optimization algorithm applies to an algebraic query of the kind discussed in Section 11.11, where the expression to be optimized represents the query applied to logical relations, which in turn are expressed in terms of physical relations. The operators assumed to appear in the query are the usual select, project, natural join, and union, plus a new operator *CHOICE*, which represents the ability of the system to choose any of the identical, replicated copies of a given relation. That is, if relations R_1, R_2, and R_3 are copies of one relation, at different nodes, and S_1 and S_2 are also copies of another relation, then we might express the join of these two relations as

$$CHOICE(R_1, R_2, R_3) \bowtie CHOICE(S_1, S_2)$$

We could, for example, save transmission cost by picking copies of R and S that were at the same node and joining them there.

One modification we shall make to expression trees is to combine nodes having the same associative and commutative binary operator into a single node, provided the nodes being combined form a tree, with no intervening nodes labeled by other operators. The three binary operators, union, natural join, and choice, are all associative and commutative, so this rule applies to each of them. A tree of nodes labeled by just one of these binary operators we shall call a *cluster*. All the nodes of a cluster will be replaced by a single node, and the children of the nodes in the cluster become children of the new node. The parent of the new node is the same as the parent of the one node in the cluster that is an ancestor of all the nodes in the cluster. The result of these operations we call a *compacted tree*.

Example 11.39: The expression tree in Figure 11.32(a) is replaced by the tree in Figure 11.32(b). We have combined the two nodes labeled ∪, since they form a cluster, and the three nodes labeled ⋈ also happen to be arranged in a tree; thus they are a cluster and are replaced by a single node. □

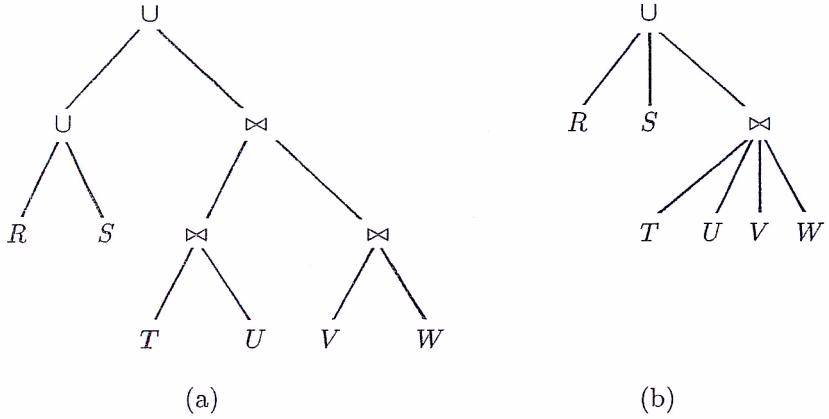


Figure 11.32 Replacement of clusters by single nodes.

Let us now enumerate the strategies for evaluation that System R* considers. We shall assume that each relation is at a single site. If a relation is replicated, then we shall give the replicas different names, and apply the *CHOICE* operator to the collection of replicas. We assume also that the query asks for a specific expression to be computed at a specific site of the network; the expression is represented as a compacted tree. The algorithm below considers all ways to evaluate the nodes of the compacted tree, taking into account

1. The various orders in which an operator like union or join of many relations can be performed as a sequence of binary steps,
2. The various sites at which each node of the tree could have its result computed, and
3. Several different ways that the join could be computed.

The Cost Function

When the algorithm considers each of the options listed above, it must evaluate the cost of these methods. The actual cost function used in System R* is quite complex; it takes account of the computation cost at a site as well as the transmission cost. We have had the flavor of the sort of analysis that is used at a single site in Section 11.2, so we shall, for simplicity, take a less detailed cost function that only accounts for the transmission cost. That is, as in Section 11.11, we shall assume that the cost of sending a message is, in appropriate units, equal to the number of tuples sent, plus a constant c_0.

Join Methods

The options for taking a union of two relations are fairly clear. We can ship one relation to the site of the other or ship them both to a third site. The costs of these three approaches are obvious, given the cost function above. The options given a *CHOICE* operator are also clear; the cost is zero for any site at which one of the relations in the choice resides, and otherwise the cost equals the cost of shipping one of the replicas to the desired site. Which replica is shipped doesn't matter, since the costs are the same, under our model.

However, with the join of two relations R and S, the set of options is not so well defined. The algorithm to be described has five options that can always be used, and two others that can be used when R and S are both at the same site. These options are the following.

1. Ship R to the site of S and compute the join there. The cost is c_0 plus the size of R.
2. Ship S to the site of R and compute the join there. The cost is c_0 plus the size of S.
3. Ship R and S to a third site and compute the join there. The cost is $2c_0$ plus the sum of the sizes of R and S.
4. Perform a semijoin $S \ltimes R$, before obtaining the relevant tuples of S and moving them to the site of R. That is, compute $\pi_{R\cap S}(R)$ at the site of R, ship these tuples to the site of S, and there compute $S \ltimes R$. Ship the result to the site of R. The cost of this method is estimated to be $2c_0 + T_R(1 + T_S/I)$, where T_R and T_S are the number of tuples in R and S, and I is the "image size," that is, the size of the projection of S onto $R \cap S$. That is, at most T_R tuples are shipped from the site of R to the site of S, and therefore $T_R T_S/I$ is an upper bound on the average number of tuples shipped back to the site of R.[40] We could estimate the cost more closely by considering the size of $\pi_{R\cap S}(R)$, but this method will only make sense when R is very small anyway.
5. There is another method similar to (4), in which the roles of R and S are reversed.

We call strategies (1)–(3) *fetches*. Strategies (4) and (5) are referred to as *lookups*.

If R and S are at the same site, there are several other strategies we shall consider.

[40] T_S is another upper bound, but this method is, in practice, only used when $T_R < I$.

6. Compute the join at the site of the two relations and leave it there. Since we charge only for transmission in our simple model, the cost of this action is zero.
7. Compute the join at the site of the two relations and ship the result to another site. The cost is the cost of shipping the result relation. Note that we could also use strategy (3), shipping the unjoined relations to a new site and joining them there. Which is preferable depends on how the size of the join compares with the sum of the sizes of R and S.

The Algorithm for Selecting an Evaluation Strategy

We can now present our simplified version of the System R* algorithm for query evaluation.

Algorithm 11.5: Selecting a Processing Strategy for a Distributed Query.

INPUT: A query in the form of a compacted tree, with operators select, project, join, union, and choice, sites for all of the argument relations, guard conditions for these relations, and a site at which the result must appear.

OUTPUT: A preferred order of application of the operators and sites at which the results of these operations should appear.

METHOD: The first stage of the algorithm is to generate all of the possible evaluation sequences, representing them as ordinary expression trees, with binary operators. We begin by pushing selections and projections as far down the tree as possible, using Algorithm 11.2. When we encounter a $CHOICE$ operator, we push the selection or projection to each child of a node labeled $CHOICE$. We also use the technique of Section 11.11 to eliminate subtrees whose guard conditions conflict with a selection and to eliminate redundant guard conditions as in Example 11.26.

We then proceed up the tree, beginning at the leaves. On visiting a node, we generate for that node a set of expression trees, each of which must be considered when finding the least cost application order.

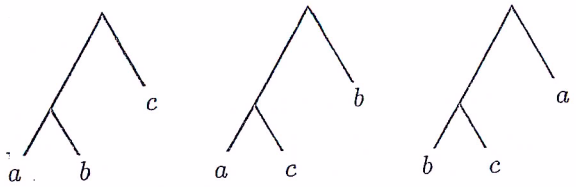
Basis: The basis, a leaf node labeled R, yields a set consisting of the one expression, R.

Inductive Step: Let n be a node of the compacted tree. Suppose the children of n, say $c_1, \ldots, c_k$, have each been processed, so we have for each child c_i a set $\mathcal{S}_i$ of expressions.

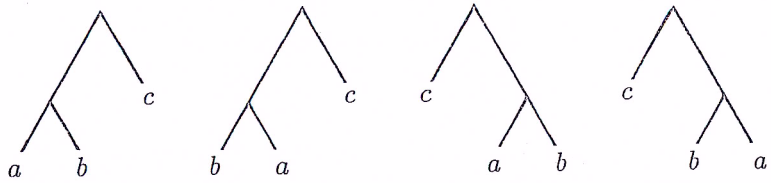
Case 1: If the operator at n is selection or projection, then $k = 1$, and we form the set of expressions for n by applying the same operator as is at n to the root of each expression tree in $\mathcal{S}_1$.

Case 2: If the operator is $CHOICE$, the set of expressions for n is the union of the $\mathcal{S}_i$'s for $i = 1, 2, \ldots, k$.

Case 3: If the operator at n is $\cup$, we consider all unordered binary trees with k leaves. For example, Figure 11.33(a) shows the three unordered trees with three leaves designated a, b, and c. Since the children of each node are unordered, the first of these could as well have been exhibited as any of the trees in Figure 11.33(b), where we have shown the two different orders of a and b, and the two different orders of c and the parent of a and b, in all possible combinations.



(a) The three unordered binary trees with three leaves.



(b) The four ordered trees that are represented by one unordered tree.

Figure 11.33 Illustration of unordered trees.

It is sufficient to consider only unordered trees; the way we group operands of a union may make a difference in the efficiency, but there can be no difference whether we compute $E_1 \cup E_2$ or $E_2 \cup E_1$. A similar remark holds for the join operator, to be discussed next. We complete construction of the expressions for node n by taking each of the unordered trees, with interior nodes labeled $\cup$, and placing at the leaf for c_i any of the expression trees in $\mathcal{S}_i$, making the replacements for the leaves in all possible ways.

Case 4: If the label of n is $\bowtie$, we proceed exactly as for a union, except that $\bowtie$ labels the interior nodes of the unordered tree.

Now we must do the second stage of the algorithm, the computation of the best evaluation strategy for each of the expression trees constructed for the root by the first phase. We now have binary trees, and again we work on each tree from the leaves up. At each node, we must compute the cost of evaluating the expression whose root is at that node, with the result left at each of the possible sites. Remember that the exact cost of operations depends not only

on the formula for charging for messages shipped, which we take to be c_0 plus the message length, but also on our estimate of the size of relations computed by the subexpressions. We shall assume that such an estimate is available, and later, we shall give an example of one possible way of making the estimate.

Basis: At a leaf node labeled R, the cost of evaluation at the site of R is zero, and the cost of evaluation at any other site is equal to the cost of shipping R, that is, c_0 plus the number of tuples in R.

Inductive Step: Consider an interior node n. If the label of n is a selection or projection, then that operation can be performed at the same site that the expression rooted at the child of n was computed. Thus, the least cost to compute n at site α equals the cost of computing the child of n at α.

Suppose the label of n is $\cup$. To find the least cost of computing n at site α, we find the minimum over all sites β and γ, of the cost of computing the children of n at β and γ, plus (if $\beta \neq \alpha$) the cost of shipping the first child's result, plus (if $\gamma \neq \alpha$) the cost of shipping the result of the second child.

If the label of n is $\bowtie$, we consider, for each site α, the cost of computing the children of n at some sites β and γ, and using any of the join methods (1)–(7) described prior to Algorithm 11.5 that apply. The least cost method is chosen to evaluate n, of course.

Having computed the cost of evaluating each of the expressions in our set at each of the sites, we find that expression with the lowest cost of evaluation for the desired output site. We must also consider the cost of computing the result at another site and shipping the result to the desired site. Having found the least cost expression, we must find the evaluation method used at each node. The proper evaluation method for a node is the one that assigned the least cost to that node. We can either recompute costs for the winning tree, to see which method is best at each node, or as we evaluate all of the possible trees, we can label each node with the proper method to use. □

Example 11.40: We shall give a simple illustration of the calculations involved in Algorithm 11.5. First, we must settle on a way of estimating the sizes of the results of operations, in particular, the join. We shall assume we know for each relation R and each subset X of the attributes in R's relation scheme, an image size, that is, the expected number of tuples in $\pi_X(R)$. As a special case, we know the expected number of tuples in R itself, since we may let $X = R$.

Suppose R and S are two relations to be joined, and let T_R and T_S be our estimates of the numbers of tuples in these relations. Let the image sizes for the set of attributes $R \cap S$ be I_R and I_S for R and S, respectively. A plausible estimate of the size of $R \bowtie S$ begins by considering the smaller of I_R and I_S; say it is I_R. Then we shall suppose that each member of $\pi_{R\cap S}(R)$ is present in the larger set $\pi_{R\cap S}(S)$. In fact, we shall assume that each tuple in R joins with an average number of tuples from S, that is T_S/I_S tuples. Thus, the size of

$R \bowtie S$ is estimated to be $T_R T_S/I_S$, or more generally, considering that either image size could be smaller,

$$\frac{T_R T_S}{max(I_R, I_S)}$$

Note that this formula agrees with (11.13) in Theorem 11.2, as long as we make the simplifying assumption that either all common attributes A of R and S satisfy the inclusion dependency $R.A \subseteq S.A$, or they all satisfy $S.A \subseteq R.A$.

Suppose that our expression requires us to take the join of three relations $P(A, B)$, $Q(B, C)$, and $R(C, D)$, which are located at three different sites α, β, and γ, respectively. As in Figure 11.33, there are three unordered trees with leaves labeled P, Q, and R, corresponding to the fact that we could take the join of any two of the three relations first. Let us assume the following constants for the problem.

1. $T_P = 10$, $T_Q = 1000$, and $T_R = 100$.
2. The image size I_{PB}, the estimated size of $\pi_B(P)$, is 10.
3. The image size I_{QB} is 20.
4. $I_{QC} = 500$.
5. $I_{RC} = 25$.
6. $c_0 = 10$.

Let us first consider joining P and Q first, and then joining the result with R. If we wish to compute the result at site α, we have two choices. We could fetch Q to α, at a cost of $c_0 + T_Q = 10 + 1000 = 1010$. Alternatively, we could examine each of the ten tuples of P and lookup the matching tuples of Q for each of them. The cost of this operation is

$$2c_0 + T_P(1 + T_Q/I_{QB}) = 20 + 10(1 + 50) = 530$$

Thus, the cost of computing $P \bowtie Q$ at site α is $min(1010, 600) = 600$.

If we wished to evaluate $P \bowtie Q$ at β instead, then there would be the two symmetric strategies, where we do a fetch or a lookup of P. The costs of these operations are, respectively, $c_0 + T_P = 20$ and $2c_0 + T_Q(1 + T_P/I_{PB}) = 2020$. We therefore prefer the fetch operation in this case, with a cost of 20.

Finally, we must evaluate the cost of computing $P \bowtie Q$ at γ. Here we must ship both relations to γ at a cost of $2c_0 + T_P + T_Q = 1030$. It is worth noting that we could do better by shipping P to β, computing the join there, and shipping the result to γ. However, there is no need to consider this approach now, since when we compute the cost of the complete expression, we shall discover that it is cheaper to evaluate $P \bowtie Q$ at β even if we want the result of that join at γ. On the other hand, if $P \bowtie Q$ were the final expression, and we wanted the result at γ, Algorithm 11.5 would tell us it is cheaper to compute the result at β and ship.

Now we must consider the cost of evaluating the entire expression,

$$(P \bowtie Q) \bowtie R$$

at each of the three sites. To begin, we must obtain our estimate of the size of $P \bowtie Q$. By the formula explained above, this size is

$$T_{PQ} = T_P T_Q / max(I_{PB}, I_{QB}) = 10 \times 1000/20 = 500$$

We can also estimate an image size for I_{PQC}, the projection of $P \bowtie Q$ onto C. Each of the 500 C-values appearing in Q is present in T_Q/I_{QC} tuples, that is, two tuples. As the ratio of I_{PB} to I_{QB} is 1/2, and we assume every B-value in P is also present in Q, it follows that half the tuples in Q will have a matching B-value in P, and will appear in the join. Thus, of the two tuples with C-value c, the probability is 3/4 that at least one of them will appear in the join. Thus the image size I_{PQC} will be approximately $\frac{3}{4} I_{QC} = 375$.

Finally, we shall need an estimate for the size of the join of all three relations. By our estimating rule, this number is

$$T_{PQR} = T_{PQ} T_R / max(I_{PQC}, I_{RC}) = 500 \times 100/375 = 133$$

Now we must consider for each of the three sites, the best way to compute $(P \bowtie Q) \bowtie R$ at that site. The choices include which site should $P \bowtie Q$ be computed at, and which of the methods (1)–(7) should be used to join $P \bowtie Q$ with R. The options are summarized in Figure 11.34.

Site of Result	Site of $P \bowtie Q$	Strategy	Cost of Joining R with $P \bowtie Q$	Cost of $P \bowtie Q$		Total Cost
α	α	Fetch R	$c_0 + T_R = 110$	530	=	640
		Lookup R	$2c_0 + T_{PQ}(1 + T_R/I_{RC}) = 2520$	530	=	3050
	β	Fetch PQ, R	$2c_0 + T_{PQ} + T_R = 620$	20	=	640
	γ	Fetch $PQ \bowtie R$	$c_0 + T_{PQR} = 143$	1030	=	1173
		Fetch PQ, R	$2c_0 + T_{PQ} + T_R = 620$	1030	=	1650
β	α	Fetch PQ, R	$2c_0 + T_{PQ} + T_R = 620$	530	=	1150
	β	Fetch R	$c_0 + T_R = 110$	20	=	130
		Lookup R	$2c_0 + T_{PQ}(1 + T_R/I_{RC}) = 2520$	20	=	2540
	γ	Fetch $PQ \bowtie R$	$c_0 + T_{PQR} = 143$	1030	=	1173
		Fetch PQ, R	$2c_0 + T_{PQ} + T_R = 620$	1030	=	1650
γ	α	Fetch PQ	$c_0 + T_{PQ} = 510$	530	=	1040
		Lookup PQ	$2c_0 + T_R(1 + T_{PQ}/I_{PQC}) = 253$	530	=	783
	β	Fetch PQ	$c_0 + T_{PQ} = 510$	20	=	530
		Lookup PQ	$2c_0 + T_R(1 + T_{PQ}/I_{PQC}) = 253$	20	=	273
	γ	None needed	0	1030	=	1030

Figure 11.34 Strategies for evaluating the join of three relations.

From Figure 11.34 it is clear that if we want the result at site β, we must compute $P \bowtie Q$ at β (by fetching P to β), then fetching R to β, with a total

cost of 130. It seems that if we want the result at α, we have two choices with the same cost, 640. For example, we could compute $P \bowtie Q$ at β (by fetching P), then fetch that result and R to α. However, we must, according to Algorithm 11.5, also consider computing the result at another site and shipping the result to α, against the cost of 640 given by Figure 11.34. In this case, computing the result at β, then shipping it with a cost of $c_0 + T_{PQR} = 143$, has a total cost of 273, which is less than the 640 given by Figure 11.34 for computation at α. Similarly, an alternative optimal strategy for computing the result at γ is to compute at β and ship, with a cost of 273, which is identical to the value given in Figure 11.34.

We are not yet done; we must consider the other two unordered trees, which involve joining P with R first, or Q with R first. The former case is not really a join but a Cartesian product, and we shall rule it out of consideration because of the size of the intermediate result, even though we said we would not consider that cost. The latter strategy, joining Q and R first, will not prove superior to what we have already, since our first step must ship a minimum of 100 tuples, that being the smaller of T_Q and T_R. □

EXERCISES

11.1: Suppose a relation $ABCD$ has a clustering index on A and nonclustering indices on the other attributes; the four indices have image sizes of 50, 10, 20, and 100, respectively. The number of tuples in the relation is 10,000, and the relation would fit on 500 blocks. Find all of the ways to evaluate the query according to Algorithm 11.1.

$$\pi_A(\sigma_{A=0 \wedge B=1 \wedge C>2 \wedge D=3}(ABCD))$$

Which method is the least costly?

* 11.2: Some of the data structures of Chapter 6 (Volume I) provide clustering indices, and some provide nonclustering indices. Indicate which of these structures yield clustering indices, which yield nonclustering indices, and which do not yield indices at all in the sense required for the System R optimization algorithm of Section 11.2. You may make reasonable assumptions about the uniformity of the data with which each structure is presented.

a) A hash table, with records stored in the buckets.
b) A hash table with records stored in a heap and pointed to by the buckets.
c) A B-tree, with records stored in the leaves.
d) A k-d-tree.
e) A B-tree whose leaves point to linked lists of records with a fixed key value (for example, as in a multilist structure).