In: Proceedings of the International Conference on Data Engineering, IEEE, April 1984, pp. 228-235.

THE FILE-ASSIGNMENT AND QUERY-PROCESSING PROBLEMS IN LOCAL MULTIACCESS NETWORKS

Benjamin W. Wah and Yao-Nan Lien
School of Electrical Engineering
Purdue University
W. Lafayette, IN 47907

ABSTRACT

In this paper the distribution-design and query-processing problems for distributed databases on local multiaccess computer networks are studied. The broadcast capability of these networks allows information to be distributed efficiently. In the distribution-design problem, the distribution of independent files are studied. With the broadcasts of updates, the distribution of multiple copies of a single file can be solved optimally in polynomial time. However, the general problem with storage-capacity constraints is still NP-hard, and an efficient branch-and-bound method is proposed. In the query-processing problem, the semi-join concept is adopted. The propagation of reduction on dependent attributes is studied. Although the problem is NP-hard, less data transmission is needed with the broadcast of information.

KEYWORDS AND PHRASES: File-allocation problem, local computer network, NP-hardness, query-processing problem, semi-join.

1. INTRODUCTION

Advances in large-scale integrated logic and communication technology coupled with the data explosion have led to the design of *distributed database management systems (DDBs)* in which data are distributed over an interconnection of processing elements. Central to the control of DDBs are the placement of files, the distributed processing of queries and the concurrency control of multiple copies.

The *distribution design* entails the splitting of a database into fragments and the distribution of these fragments on the network so that the system requirements are satisfied [CER83]. When each fragment is considered as a file, the distribution-design problem becomes the *file-allocation problem (FAP)* [WAH84,DOW82,CHU69]. A special case is the *simple file-allocation problem (SFAP)* in which a single file is considered, and the effects of queries, updates and data storage are represented as costs on the system [RAM83]. The objective is to minimize the overall operating costs.

A *query* is an access or a sequence of accesses on files in the database. In general, a query may originate from a program, and the files accessed may be distributed in the network. An effective arrangement of local data processing and data transmissions must be derived so that the amount of data transfer or the response time of processing is minimized [YAO79]. This arrangement is known as a *distribution strategy* for a query. The problem is usually referred to as the *distributed query-processing problem (DQP)*.

Concurrency control is a control mechanism which maintains the integrity of redundant information in a DDB [BER81b]. This control is usually complicated because (1) users may access information stored in many computers; (2) communication delays prohibit instantaneous distribution of status information; and (3) recovery must be possible when a computer or the network fail.

The design of DDBs on local computer networks (LCN) is becoming important with the advent of office-information systems and CAD/CAM applications. These networks are characterized by a diameter no more than a few kilometers, and the data rate usually exceeds one megabits per second [TAN81]. One important topology is the multiaccess bus in which any information sent on the bus is available to all the listening stations. This property is especially useful in distributed database management when multiple copies of the same file have to be updated. In this case, one broadcast is sufficient to request locks on all redundant copies of information, followed by another broadcast to update all the copies. The serializability problem for concurrency control in general distributed databases [BER81b] is straightforward here because every transaction is known to all the nodes, and all transactions are carried out in the same order.

In this paper FAP and DQP on local computer networks with a multiaccess bus are studied. It is assumed that the predominant overhead lies in data transmission. Since the transmission costs for all remote accesses and updates are constant, the SFAP is greatly simplified (Section 2). The general FAP is still NP-hard, and a branch-and-bound method is proposed (Section 3). The broadcast of information also reduces redundant information transfer in query processing (Section 4).

2. SIMPLE FILE ALLOCATION ON LOCAL MULTIACCESS NETWORKS

The SFAP considers the allocation of a single file in which the effects of queries, updates and data storage are represented as costs [RAM83]. It has been shown to be NP-hard [ESW74] which means that the computation time for all known optimal algorithms for this class of problems increases at least exponentially with the problem size. Numerous exhaustive algorithms and suboptimal heuristics have been studied [WAH84,DOW82]. In this section a formulation of the SFAP on local multiaccess networks is proposed. Due to the broadcast capability, the problem can be solved in polynomial time with respect to the number of nodes.

The general notations used in the formulation are:

N - number of sites in the system
F - set of files in the database, $|F| = m$
I^f - index set of sites with a copy of file f
λ_j^f - query load originating at site j for file f per unit time

This research was partially supported by National Science Foundation Grant ECS81-05968 and by CIDMAC, a research unit of Purdue University, sponsored by Purdue, Cincinnati Milicron Corporation, Control Data Corporation, Cummins Engine Company, Ransburg Corporation and TRW.

ϕ_j - update load originating at site j for file f per unit time

d - query cost per unit size of query from any site to any other site

d' - update cost per unit size of update from any site to all other sites

σ_k^f - storage cost per unit time of file f at site k

L^f - length of file f

S_k - storage capacity at site k

$$Y_k^f = \begin{cases} 1 & \text{if file f exists at site k} \\ 0 & \text{otherwise} \end{cases}$$

Since the SFAP is defined with respect to a single file, the index f is dropped here. The cost function is:

$$C(I) = \sum_{j=1}^{N} \lambda_j \cdot d \cdot (1-Y_j) + \sum_{j=1}^{N} \phi_j d' \left[1 - Y_j \prod_{\substack{k=1 \\ k \neq j}}^{N} (1-Y_k) \right] + \sum_{k=1}^{N} \sigma_k Y_k \quad (1)$$

The first and last terms in the above equation represent the query and storage costs respectively. The second term accounts for the update costs. When multiple copies of the file exist in the system, each unit of update transaction incurs a constant cost of d'. However, when only one copy of the file exists, updates originating from the site with the copy do not have to be broadcast and will not incur any cost.

The problem can be formulated as an integer program with the cost function given by Eq. 1 and the constraints that at least one copy exists and Y_k is either 0 or 1. However, the problem can be simplified by considering the single-copy and the multi-copy cases separately.

In the single-copy case, the cost function is:

$$C(I) = \sum_{j=1}^{N} [(\lambda_j d + \phi_j d') \cdot (1-Y_j) + \sigma_j Y_j] \quad (2)$$

$$= \sum_{j=1}^{N} (\lambda_j d + \phi_j d') + \sum_{j=1}^{N} (\sigma_j - \lambda_j d - \phi_j d') Y_j$$

The cost of not placing a copy at site j is $\lambda_j d + \phi_j d'$ while the cost of placing a copy there is σ_j. Thus the file should be placed at site j if the cost difference $(\sigma_j - \lambda_j d - \phi_j d')$ is minimum. The optimal allocation can, thus, be found in O(N) time.

In the multi-copy case, at least two copies must be allocated. The cost function is:

$$C(I) = \sum_{j=1}^{N} [\lambda_j d(1-Y_j) + \phi_j d'] + \sum_{k=1}^{N} \sigma_k Y_k \quad (3)$$

$$= \sum_{j=1}^{N} (\lambda_j d + \phi_j d') + \sum_{j=1}^{N} (\sigma_j - \lambda_j d) Y_j$$

The first term in the above equation is constant. The optimal allocation can be found by allocating a copy at site j if $\sigma_j - \lambda_j d$ is negative which will decrease the overall cost, C(I). In case that none or one of the cost differences are negative, two copies with the minimum cost differences are selected. It is obvious that the optimal allocation can also be found in O(N) time.

A global optimum can be obtained by comparing the costs under the single-copy and multi-copy cases. The problem is optimally solvable in polynomial time. An example illustrating the algorithm is shown in Figure 1.

The SFAP under an availability constraint can also be solved easily. Assuming that the failure rates of sites are identical and that the network is reliable, availability can be maintained by a minimum number of copies of the file. The locations for these copies can be determined efficiently.

On the other hand, the SFAP becomes NP-hard when an average-delay constraint is imposed. Given that

Given
number of nodes: n = 4;
query rates: $\lambda_* = [6\ 7\ 4\ 5]$;
update rates: $\phi_* = [3\ 6\ 2\ 4]$;
storage costs: $\sigma_* = [3\ 5\ 2\ 5]$;
unit query/update cost: d = d' = 1.

Single-copy case:
$\sigma_* - \lambda_* d - \phi_* d' = [-6\ -8\ -4\ -4]$;
$[Y_*] = [0\ 1\ 0\ 0]$;
cost = (6+3)+(5)+(4+2)+(5+4) = 29.

Multi-copy case:
$\sigma_* - \lambda_* d = [-3\ -2\ -2\ 0]$;
$[Y_*] = [1\ 1\ 1\ 0]$;
cost = (3+3)+(6+5)+(2+2)+(5+4) = 30.

Solution: Single copy placed at at node 2 has minimal cost.

Figure 1. Example to illustrate simple file allocation on multiaccess networks.

the average delay on the multiaccess bus is constant (=c), the average delay for all the queries must satisfy:

$$\sum_{j=1}^{N} \lambda_j c(1-Y_j) \leq D \quad (4)$$

The optimization problem on the locations of copies can be solved in the single-copy and multi-copy cases. The single-copy case is solvable in O(N) time. The cost formula for the multi-copy case (Eq. 3) can be rewritten as:

$$C(I) = \sum_{j=1}^{N} (\phi_j d' + \sigma_j) + \sum_{j=1}^{N} (\lambda_j d - \sigma_j)(1-Y_j)$$

Since the first term on the RHS is constant, the optimization problem can be formulated as:

$$\text{maximize } C(I) = \sum_{j=1}^{N} (\sigma_j - \lambda_j d)(1-Y_j) \quad (5)$$

$$\text{subject to } \sum_{j=1}^{N} \lambda_j (1-Y_j) \leq \frac{D}{c} \text{ and}$$

$$1-Y_j = 0 \text{ or } 1$$

The problem is reducible from the 0-1 knapsack problem*. In fact, by a transformation of variables, $Z_j = 1-Y_j$, the problem is the 0-1 knapsack problem itself with N objects in which the profit and weight of the j-th object are $(\sigma_j - \lambda_j d)$ and λ_j respectively. However, the profits may be negative here, and the algorithms to solve the knapsack problem have to be modified slightly in order to accommodate this fact. The problem is not strongly NP-hard [GAR79] and is solvable by dynamic programming [NEM69] or fully polynomial-time approximate schemes [GAR79].

3. GENERAL FAP WITH STORAGE-CAPACITY CONSTRAINTS

The general FAP considers the allocation of multiple files under such design requirements as delay, storage capacity, parallelism and availability. Each query is assumed to access a single file so that all the files are independently accessed. Specifically, the storage capacity of sites and the average access delays are considered in the formulation.

*Given a finite set U of m objects and a knapsack of size B. For each $u \in U$, there are a size $s(u) \in Z^+$ and a profit $p(u) \in Z^+$. Is there a subset $U' \subseteq U$ that maximizes $\sum_{u \in U'} p(u)$ such that $\sum_{u \in U'} s(u) \leq B$?

Using the notations defined in the last section, the problem with storage-capacity constraints can be formulated as:

minimize $C(I^1, ..., I^{|F|})$ (6)

$$= \sum_{f,j} \lambda_j^f d(1-Y_j^f) + \sum_{f,j} \phi_j^f d' \left[1 - Y_j^f \prod_{k \neq j} (1-Y_k^f)\right] + \sum_{f,j} \sigma_j^f Y_j^f$$

subject to:

(a) each site can contain at most one copy of each file,
$Y_j^f = 0$ or $1 \qquad j = 1, ..., N; f \in F$

(b) at least one copy of each file exists in the system,
$\sum_{j=1}^{N} Y_j^f \geq 1, \quad f \in F$

(c) the storage capacity at each site is not exceeded,
$\sum_{f \in F} L^f Y_j^f \leq S_j \quad j = 1, ..., N$

The above formulation is non-linear. However, it can be linearized easily by using different index variables [GEO72]. Moreover, the problem is NP-hard. This is shown in the following theorem.

Theorem: The FAP on LCN with capacity constraints is NP-hard.

Proof: We show that 0-1 knapsack problem reduces to FAP in polynomial time. Given an instance of the 0-1 knapsack problem, an instance of the FAP can be formed with following parameters: $N = 2$; $F = U$; $L^f = s(f), f \in F$; $S_1 = \sum_{f \in F} L^f$; $S_2 = B$; $\sigma_1^f = 0, f \in F$; $\lambda_1^f = a$ large constant, $f \in F$; ϕ_1^f, λ_2^f and σ_2^f are chosen such that $\lambda_2^f - \phi_1^f d' - \sigma_2^f = p(f)$, $f \in F$. Basically, the first site is large enough to hold a copy of all the files in F. The query and storage costs are chosen so that a copy of all the files are allocated at site 1. Therefore, the problem becomes the packing of the second knapsack which is the standard knapsack problem. ■

Although the problem resembles the packing of m·n copies into m knapsacks, there is a subtle difference. It is observed that the profit of allocating the first copy of file f to site j is $(\lambda_j^f d + \phi_j^f d' - \sigma_j^f)$. The profit of allocating the second copy of file f to site k, $k \neq j$ is $(\lambda_k^f d - \phi_j^f d' - \sigma_k^f)$. There is an extra term $\phi_j^f d'$ in the profit of the second copy because it offsets the additional profit incurred when the first copy is allocated. The profit of allocating other copies of file f to site $\ell \neq j \neq k$ is $(\lambda_\ell^f d - \sigma_\ell^f)$. Note that the profits may be negative. Due to the above facts and since more than one copy must exist in the system, it is impossible to decompose the problem and solve it as multiple independent knapsack problems. However, the problem is decomposable if at least two copies have been allocated in the system. In this case the overall profit can be maximized by optimizing independently the profit of each site with the remaining capacity. An optimal algorithm, therefore, consists of enumerating the allocations of the first two copies of each file in the database, and solving N knapsack problems for each combination. The structure of the state-space tree is as follows:

(1) The root is at level 0.
(2) In the first m levels, the first copy of all the files are allocated. Level f, $1 \leq f \leq m$, represents the allocation of file f. Each file can be allocated to one of the N sites. Thus the degree of the tree in the first m levels is N. If a file cannot fit in a site, the subtree representing this allocation is considered infeasible and terminated.
(3) The allocation of the second copy of each file is carried out in the $(m+1)$-st to 2m-th levels of the tree. Since the second copy is not essential and must not be allocated to the site containing the first copy, the degree of branching at each level is still N. If a file cannot fit in a site, the subtree representing this allocation is considered infeasible and terminated. Further, if an allocation of the second copy at site k results in negative profit $(\lambda_k^f d - \sigma_k^f < 0)$ then the second copy should not be allocated there, and the subtree is also terminated.
(4) For all active nodes left in the state-space tree at level $2m+1$, the allocation of the remaining copies is solved as N single-knapsack problems. For a particular site, the files to be considered for packing into the remaining capacity must be one that has not been allocated there and one whose second copy is allocated. The algorithm for solving the conventional 0-1 knapsack problem has to be modified because the profits may be negative.

Bounding criteria similar to those used in branch-and-bound algorithms can be developed. In particular, lower and upper bounds can be computed for each node in the state-space tree. The upper bound on profit of a node can be calculated by a linear program or a greedy algorithm without the integrality constraints. In computing the lower bound, a feasible allocation has to be found. Due to the constraints imposed, the lower bounds may be difficult to evaluate. A global incumbent contains the maximum of the set of feasible solutions. If a node in the state-space tree has an upper bound smaller than the incumbent solution, the subtree originating from this node is pruned because it cannot possibly lead to an optimal solution. An example illustrating the allocation of two files in a system with two sites using depth-first search is shown in Figure 2. The worst case complexity of the algorithm is mN^{2m+1} which is much better than exhaustive enumeration.

4. QUERY PROCESSING IN LOCAL MULTIACCESS NETWORKS

Nearly all the studies on distributed query processing were developed for relational databases on a non-broadcast system with identical transmission cost per unit data between any two points in the network. A relational query performs restrictions, projections and joins [COD70]. Since the cost of inter-node join operations is expensive, most of the studies were focused on reducing this cost.

Wong presented a heuristic for query processing [WON77]. An initial feasible strategy is selected, then a "hill-climbing" heuristic is applied recursively until no further cost improvements can be discovered.

Bernstein and Chiu [BER81a] incorporated semi-joins into DQP strategies and showed that they were very effective as compared to a full join. Basically, in joining two distributed relations, one of the joining relations can be shrunk in size by deleting those tuples that cannot play a role in the join. This is done by sending values of the joining attribute of the other relation to the first relation and performing a join there. The join is completed by sending the reduced form of the first relation to the second one.

A comprehensive approach to query-processing optimization has been developed by Yao and Hevner [HEV79,YAO79,APE83]. In their approach, the semi-join concept was adopted for minimizing response time and total time. Yu et al. reduced the transmission cost further by incorporating complement transmissions into

Cost of query, update, and storage

	file 1		file 2	
	site 1	site 2	site 1	site 2
$\lambda_j^I d$	8	9	6	7
$\varphi_j^I d'$	1	1	1	1
σ_k^I	4	6	4	6

Storage limit of sites

site 1	site 2
5	6

Length of files

file 1	file 2
3	4

Profits of first and second copies

	file 1		file 2	
	site 1	site 2	site 1	site 2
First copy	5	4	3	2
Second copy	3	2	1	0

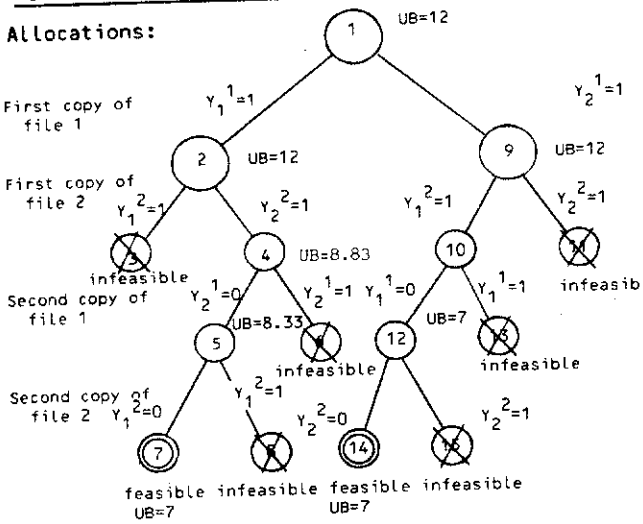


Figure 2. Example to illustrate general file allocation with storage constraints (N=2, |F|=2)

semi-joins [YU82]. During semi-join transmissions, the complement set of attribute values are sent if its cardinality is less than that of the original attribute. The transmission cost can be reduced; however, there is no efficient algorithm for determining the optimal query-processing sequence even for simple queries in which every requested relation has one common attribute. On the other hand, traditional data compression techniques can be used in semi-join transmissions [GOU81]. For instance, a bit vector indicating the absence or presence of a value can be sent instead of the attribute values if the bit-vector size, which is $\log_2$(cardinality of possible values of the attribute), is smaller than the attribute size.

Chu and Hurley developed a unified approach in considering both the local-processing and transmission costs in their algorithm [CHU82]. A query-tree model for selecting the transmission sequence and the sites for executing a set of subqueries in order to minimize the operating cost was proposed.

The design of query-processing strategies on systems with multiaccess networks is different from that on systems with general networks because data can be broadcast in parallel. Less data have to be transmitted, and the processing order may be different. In this paper we focus on deriving a semi-join schedule which minimizes the amount of data transmission for inter-node joins. It is assumed that all predicates are expressed as conjunctive equi-joins.

4.1 PROCESS OF JOINS

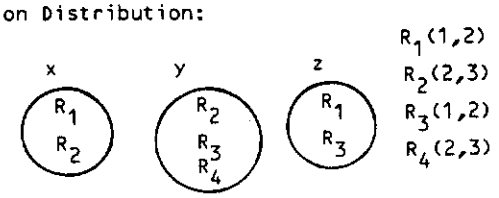
The process of multi-relation joins is carried out in five phases: materialization, local semi-joins, scheduling, global semi-joins and relation transmissions.

Materialization is the identification of one or more copies of each relation that is referenced by the query. The materialization is ***non-redundant*** if only one copy of each relation is identified; otherwise it is ***redundant***. Non-redundant materialization is assumed in most query-processing strategies in order to simplify the scheduling. However, redundant materialization usually results in better performance because the size of any attribute after performing a semi-join is less than or equal to the original size. This guarantees that the transmission cost will not increase when duplicated copies are used. Further, the amount of data transmission of any DQP schedule as obtained from a non-redundant materialization is no less than that of the same schedule for any redundant materialization, provided that the non-redundant materialization is a subset of the redundant materialization. Due to page limitation, only non-redundant materialization is studied in this paper. Results on redundant materialization will be presented in a future paper.

Assuming that a unique copy of each relation is identified, local restrictions and projections are first performed. Local joins are not performed because the size of joined relations may be very large. Instead, all the local semi-joins are performed, and statistical information on attributes are either estimated or collected by the scheduler for scheduling the global semi-joins. The problem of scheduling local semi-joins is very similar to the problem of scheduling global semi-joins because each local semi-join can be considered as a local broadcast with zero cost. The algorithm developed in this paper can be adapted to estimate statistical information at the end of the local semi-join phase.

In the global semi-join phase, projected values of attributes in each site are scheduled for broadcast in sequence. It is assumed that all the distinct attributes at a site are referenced through a virtual relation. Define the ***complement attribute*** as the set of values absent from a given attribute. All complement attributes before the global semi-join phase are called the ***initial complement attributes***. As semi-joins are carried out, the size of an attribute is going to decrease, and the size of the corresponding complement attribute is going to increase. This means that the size of an updated complement attribute is larger than the size of the corresponding initial complement attribute. In scheduling data for broadcast in a semi-join, if the cardinality of an updated attribute is smaller than that of the initial complement attribute, it is more efficient to broadcast the updated attribute itself (called ***normal broadcast***). Otherwise, a ***complement broadcast*** is selected which uses the initial complement attribute for broadcasting. After each broadcast, all relations with the common joining attribute are reduced in size. After the global semi-join phase, the fragments of identified relations are sent sequentially over the network

Relation Distribution:



Query:

$(R_1.1=R_3.1)$ AND $(R_1.2=R_2.2=R_3.2=R_4.2)$ AND $(R_2.3=R_4.3)$

Figure 3a. Data distribution before materialization

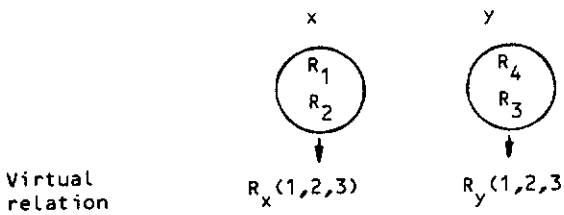


Figure 3b. Materialization and virtual relations after local semi-joins

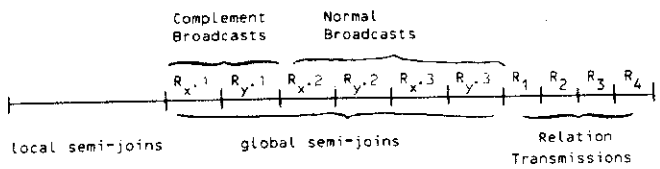


Figure 3c. An example schedule

to the result site, and the complete join is performed there.

For example, the processing of a query with predicate "$(R_1.1=R_3.1)$ AND $(R_1.2=R_2.2=R_3.2=R_4.2)$ AND $(R_2.3=R_4.3)$" is illustrated in Figure 3. The distributions of relations in sites x, y and z are shown in Figure 3a. The materialization identifies a unique copy of each relation (Figure 3b). All the distinct attributes at a site are grouped together and referenced through a virtual relation. The schedule is designed after local semi-joins, and an example schedule is shown in Figure 3c. Complement broadcast of attribute 1 in site x is first carried out. R_3 and R_4 in site y perform semi-joins simultaneously, and the statistical information on their attributes are changed. Other attributes are broadcast in sequence. Finally, the reduced relations are sent to the result site.

4.2 PARAMETERS

In this section parameters similar to those used in other studies are defined.

X - set of sites containing relations;
R^x - set of relations in site $x \in X$;
R - set of virtual relations, one at each site;
$r_{i,x}$ - number of records in R_i, $R_i \in R^x$;
$s_{i,x}$ - size of R_i, $R_i \in R^x$;
M - set of distinct joining attributes of R;
M_i - set of joining attributes in R_i, $R_i \in R$, $M_i \subseteq M$, $m_i = |M_i|$;
$d_{i,j}$ - attribute j of virtual relation i, $R_i \in R$, $j \in M_i$.

For convenience, $\bigcup_{R_i \in R} d_{i,j}$ is defined as ***domain of attribute*** *j* (domain j in short). Domain j is ***functionally dependent*** on domain i if for any relation containing domains i and j, values in attribute i are mapped in a one-to-one fashion onto values in attribute j [COD70]. For domain j:

V_j - number of distinct values in domain j before the global semi-join phase;
w_j - size of data item in domain j.

For each attribute $d_{i,j}$, $j \in M_i$, $R_i \in R$;

$u_{i,j}$ - number of distinct values that attribute $d_{i,j}$ currently has;

$p_{i,j}$ - attribute selectivity, $p_{i,j} = \frac{u_{i,j}}{V_j}$, $(0 \le p_{i,j} \le 1)$;

$w_{i,j}$ - size of data item in the attribute, $s_{i,x} = r_{i,x} \times \sum_{j \in M_i} w_{i,j}$;

$b_{i,j}$ - projected size of the attribute with no duplicated values, $b_{i,j} = u_{i,j} \times w_{i,j}$.

When an attribute is broadcast over the network, the selectivities of other attributes are affected. Sometimes, the effect has been accounted for earlier, and should not be counted again in the current broadcast. As an example, suppose there are three relations with attributes 1 and 2 at sites x, y and z. When $R_x.1$ is broadcast, $R_y.1$ and $R_z.1$ are semi-joined. This in turn will affect $R_y.2$ and $R_z.2$. Later, when $R_y.2$ is broadcast, the effect of $R_x.1$ should not propagate to $R_z.2$ because this has already been accounted for in the earlier broadcast of $R_x.1$. Another effect that must be discarded is the reduction effect on the relation containing the broadcast attribute. For example, after the broadcast of $R_x.1$ which affects R_y and R_z, this reduction effect should not propagate back to R_x when any of the attributes in R_y or R_z is broadcast in the future. This accumulation of selectivities for attributes that have already been broadcast is included in the following variable.

$COM_{x_1,x_2,j}$ - accumulation of effects that have affected the sizes of $d_{x_1,j}$ and $d_{x_2,j}$ simultaneously, or when R_{x_2} contains the currently broadcast attribute which affects the size of $d_{x_1,j}$ so that when $d_{x_1,j}$ is broadcast in the future, this effect should not be accounted for in $d_{x_2,j}$ ($COM_{x_1,x_2,j}$s are initialized to 1).

For each attribute d_{i_2,j_2}, $i_2 \in R$, $j_2 \in M_{i_2}$, a set of values are defined:

SR - size reduction indices = $\{z_{i_1,j_1,i_2,j_2} \mid R_{i_1}, R_{i_2} \in R, j_1,j_2 \in M\}$ with the following properties:
(1) if $i_1 = i_2$, $z_{i_1,j_1,i_2,j_2} = 0$;
(2) if $i_1 \neq i_2$ and $j_1 = j_2$, $z_{i_1,j_1,i_2,j_2} = 1$;
(3) if $d_{i_2,j_1} = \emptyset$, $z_{i_1,j_1,i_2,j_2} = 0$;
(4) otherwise,

$z_{i_1,j_1,i_2,j_2} =$
- 0 if j_1, j_2 are functionally independent
- 1 if j_1 is functionally dependent on j_2
- 2 if j_2 is functionally dependent on j_1

The indices SR indicate the relationships of attributes. A value of zero indicates no relationship; a value of one indicates strong relationship; and a value of two indicates partial relationship. If attribute d_{i_1,j_1} with current selectivity p_{i_1,j_1} is broadcast, the sizes and selectivities of attributes in the same relation are not reduced by this broadcast. In this case, z_{i_1,j_1,i_1,j_2} is set to zero. For attribute d_{i_2,j_1}, $i_2 \neq i_1$, that has to be semi-joined with d_{i_1,j_1}, the attribute size is reduced by a factor of $\frac{p_{i_1,j_1}}{COM_{i_1,i_2,j_1}}$. In this case, z_{i_1,j_1,i_2,j_1} is defined as 1. For attribute $j_2 \neq j_1$ in rela-

tion $R_{i_2} \neq R_{i_1}$ such that $d_{i_2,j_1} = \emptyset$, z_{i_1,j_1,i_2,j_2} is set to zero since R_{i_2} cannot be joined to R_{i_1} on domain j_1. Lastly, if d_{i_2,j_1} is reduced by the broadcast, d_{i_2,j_2} will also be affected since the size of relation R_{i_2} is reduced. This change is difficult to estimate and has to be classified for different functional dependencies.

Functional dependencies between any pair of domains j_1 and j_2 in relation R_{i_2} can be classified into three cases. Partial dependency is ignored since the reduction effects are too difficult to estimate.

(1) Domain j_1 and j_2 are functionally independent: Every value in d_{i_2,j_1} may be associated with any value in d_{i_2,j_2}. Then it is very likely that the reduction on d_{i_2,j_1} has insignificant effects on d_{i_2,j_2}. z_{i_2,j_1,i_2,j_2} is, thus, set to zero.

(2) Domain j_1 is functionally dependent on domain j_2: If there is a value in d_{i_2,j_2}, it should have one and only one corresponding value in d_{i_2,j_1}. Once a value in d_{i_2,j_1} is eliminated during a semi-join, one or more values in d_{i_2,j_2} should be eliminated as well. It is assumed that the values in d_{i_2,j_2} are randomly distributed, so the size of d_{i_2,j_2} is reduced by the same factor as the d_{i_2,j_1}. Therefore, z_{i_2,j_1,i_2,j_2} is set to one.

(3) Domain j_2 is functionally dependent on domain j_1: There are one or more values in d_{i_2,j_1} which can determine the value of d_{i_2,j_2}. Once a value in d_{i_2,j_1} is eliminated, the corresponding value v in d_{i_2,j_2} is also eliminated if there is no other value in d_{i_2,j_1} that corresponds to v in d_{i_2,j_2}. z_{i_2,j_1,i_2,j_2} is set to two here. An urn model is used to estimate the remaining cardinality of d_{i_2,j_2}. Suppose each distinct value in d_{i_2,j_2} is treated as an urn, and each distinct value in d_{i_2,j_1} is considered as a ball. Assuming that the functional dependency is well-defined, that is, all the urns are non-empty. After the semi-join, the size of d_{i_2,j_1} is reduced from u_{i_2,j_1} to u'_{i_2,j_1}. This is equivalent to drawing $u_{i_2,j_1} - u'_{i_2,j_1}$ balls out from the urns. The number of urns that become empty are the number of values eliminated from d_{i_2,j_2}.

Assume that r_1 balls are to be randomly distributed into n urns and that the n urns are non-empty. This can be done by first distributing n balls, one to each urn, and distributing the remaining $r_1 - n$ balls randomly. Consider a particular urn, the probability that this urn has k, $k = 1, \ldots, r_1 - n + 1$, balls (including the one distributed earlier) is:

$$\Pr(k) = \binom{r_1 - n}{k-1} \left(\frac{1}{n}\right)^{k-1} \left(\frac{n-1}{n}\right)^{r_1 - n - k + 1} \tag{7}$$

The probability of selecting k balls from this urn conditioned on the fact that no more than k balls can be selected and that the balls are selected in x, $x \geq k$, trials is:

$$\Pr\left[\begin{matrix}\text{urn is} \\ \text{empty}\end{matrix} \middle| \begin{matrix}\text{k balls,} \\ \text{x trials}\end{matrix}\right] = \binom{x}{k} \left(\frac{1}{n}\right)^{k} \left(\frac{n-1}{n}\right)^{x-k} \tag{8}$$

Assuming the $r_2 \leq r_1$ balls are removed from the urns, the probability emptying the urn in the x-th trial given that there are k events in x trials and $x \leq r_2$ is:

$$\Pr(X=x \mid k, x \leq r_2) = \frac{\binom{x-1}{k-1}}{\binom{x}{k}} \Bigg/ \sum_{q=k}^{r_2} \frac{\binom{q-1}{k-1}}{\binom{q}{k}} \tag{9}$$

Therefore,

$$\Pr\left[\begin{matrix}\text{urn is} \\ \text{empty}\end{matrix}\right] = \sum_{k=0}^{n} \left\{ \sum_{x=k}^{r_2} \left[\begin{matrix}\text{urn is} \\ \text{empty}\end{matrix} \middle| \begin{matrix}\text{k balls,} \\ x \leq r_2\end{matrix}\right] \Pr\left[\begin{matrix}X=x \mid \\ k, x \leq r_2\end{matrix}\right] \right\} \Pr(k) \tag{10}$$

The expected number of empty urns is, thus, n times the probability in Eq. 10.

In summary, after attribute d_{i_1,j_1} is broadcast, attribute d_{i_2,j_1} is reduced in size from u_{i_2,j_1} to u'_{i_2,j_1}. The size and selectivity of attribute d_{i_2,j_2} will be reduced in the following way:

$$b'_{i_2,j_2} = \begin{cases} b_{i_2,j_2} & \text{if } z_{i_1,j_1,i_2,j_2} = 0 \\ b_{i_2,j_2} * p_{i_1,j_1} / COM_{i_1,i_2,j_1} & \text{if } z_{i_1,j_1,i_2,j_2} = 1 \\ b_{i_2,j_2} - w_{j_2} * u_{i_2,j_2} * \Pr\left[\begin{matrix}\text{urn is} \\ \text{empty}\end{matrix}\right] & \text{if } z_{i_1,j_1,i_2,j_2} = 2 \end{cases} \tag{11}$$

In calculating Pr(urn is empty), the following values are used: $n = u_{i_2,j_2}$, $r_1 = u_{i_2,j_1}$ and $r_2 = u_{i_2,j_1} - u'_{i_2,j_1}$.

$$s'_{i,i_2} = s_{i,i_2} * \frac{b_{i_2,j_2}}{b_{i_2,j_2}} \quad \text{if } i_2 \neq i_1,\ R_i \in R^{i_2} \tag{12}$$

$$p'_{i_2,j_2} = p_{i_2,j_2} * \frac{b'_{i_2,j_2}}{b_{i_2,j_2}} \tag{13}$$

For all $i_2, i_3 \in R$, $j \in M$, if $z_{i_1,j_1,i_2,j} * z_{i_1,j_1,i_3,j} \neq 0$ or ($i_3 = i_1$ and $z_{i_1,j_1,i_2,j} = 1$), then

$$COM'_{i_2,i_3,j} = COM_{i_2,i_3,j} * \frac{b'_{i_2,j_2}}{b_{i_2,j_2}} \tag{14}$$

4.3 SCHEDULE DESIGN

In designing a schedule for query processing, it is assumed that the cost of local processing is negligible. This is especially true in multiaccess networks in which the transmission speed is relatively slow. The cost of processing a query is, thus, made up of the cost of semi-joins and the cost of fragment transmission to the result site. The cost of broadcast in the k-th step of global semi-joins can be formulated recursively as follows:

$$C_k(M_k) = \min_{d_{i,j} \in W} \left\{ \beta_{i,j} + C_{k+1}(M_{k+1}(M_k, d_{i,j})) \right\} \tag{15}$$

$k = 1, \ldots, K$, and K is the total number of semi-join broadcasts.

$$C_{K+1} = \sum_{x} \sum_{R_i \in R^x} s_{i,x} \quad \begin{matrix}\text{(boundary condition:} \\ \text{cost of final relation} \\ \text{transmissions)}\end{matrix} \tag{16}$$

where

$$\beta_{i,j} = \begin{cases} b_{i,j} & \text{if } b_{i,j} \leq b^0_{i,j} \text{ (size of initial complement)} \\ b^0_{i,j} & \text{otherwise} \end{cases} \tag{17}$$

W: set of attributes waiting for broadcast

M_k: database state in Step k

$$= \left\{ \{b_{i,j}\}, \{p_{i,j}\}, \{COM_{i,i',j}\}, \{s_i\}, W \,\middle|\, i, i' \in R, j \in M \right\}$$

$M_{k+1}(M_k, d_{i,j})$: database state in Step $k+1$ with previous state at M_k and the broadcast of $d_{i,j}$;
$W = W - \{d_{i,j}\}$;
all other parameters are updated as shown in Section 4.2.

M_1: (initial state) $W = \{d_{i,j} \mid R_i \in R, j \in M\}$;
$COM_{i,i',j} = 1$, $i, i' \in R$, $j \in M$;
$b_{i,j}$, $p_{i,j}$ and s_i, $R_i \in R$, $j \in M$ are estimated after the local semi-joins.

The cost of final relation broadcasts to the result site is the total size of the remaining fragments of relations (Eq. 16). This cost is dependent on the order of broadcasts because some reduction effects have not been considered during semi-joins, and the sizes of fragments can be further reduced. Bernstein and Chiu had pointed out that semi-joins may not execute a join completely [BER81a]. However, these effects are not predominant and are difficult to estimate, and hence are excluded from consideration.

In designing a schedule for query processing, it is noted that the problem is NP-hard. Moreover, the problem of deciding whether an attribute should be sent in complement form cannot be made before scheduling because the decision depends on the dynamic values of size and selectivity. However, the following property on complement transmission is useful. Define a *C-N ordering schedule* as a DQP schedule in which all the complement broadcasts are done before the normal broadcasts. Yu has shown that for any schedule S not in C-N ordering, there exist a C-N ordering schedule with all normal broadcasts in the same order as in S and no increase in cost [YU82]. This property reduces the search to C-N ordering schedules only, and the search complexity is significantly reduced.

In the remaining part of this section, a simple and efficient heuristic for query processing is shown. The algorithm is a greedy algorithm which repeatedly broadcast the attribute with the minimum size.

Query Processing Heuristic

(1) $I = \{$joining attributes$\}$;
(2) **while** $I \neq \emptyset$ **do** [
(3) /* Suppose $d_{i,j}$ is chosen to be broadcast, define the size to be a function of $b_{i,j}$, the total size reduction in attribute $d_{*,j}$ $(= \sum_{k \in R} b_{k,j})$ before and after the broadcast, and the total size reduction in attribute $d_{*,\ell}$, $\ell \neq j$, $(= \sum_{\substack{k \in R \\ \ell \neq j}} b_{k,\ell})$ before and after the broadcast. */
select $d_{i,j} \in I$, such that the size is minimum;
(4) set $d_{i,j}$ to be the broadcast attribute;
modify all the related sizes and selectivities in the database;
(5) $I = I - \{d_{i,j}\}$]

The size computation in Step 3 depends on the total size reduction before and after the broadcast. The size can be the absolute size or the projected size (selectivity × absolute size) of an attribute. We illustrate the heuristic in the following example.

Referring to Figure 3, four distributed relations R_1, R_2, R_3 and R_4 are referenced in a query. Statistics on the relations are shown in Figure 4a. The first three steps of heuristic scheduling are shown in Figure 4b. The broadcast sequence and the total costs using absolute size measure are shown in Table 1.

In broadcasting an attribute, it is noted that its size and its effects on other attributes decrease as it is delayed in broadcast. A good heuristic should, therefore, schedule an attribute with small size and ratio of selectivity to COM as early as possible so that the broadcast can reduce a larger number of the bigger attributes. Moreover, since the later broadcasts are usually very small in

Dependencies: domain 2 functionally depends on domain 3

Statistics after local joins:

Relation Sizes	S_1	S_2	S_3	S_4
	10000	20000	30000	40000

Attribute sizes and selectivities

	b_{i1}	p_{i1}	b_{i2}	p_{i2}	b_{i3}	p_{i3}
R_x	900	0.9	250	0.25	1000	0.1
R_y	800	0.8	700	0.7	9000	0.9
v_j	1000		1000		10000	

Sizes of Initial Complement

	j=1	j=2	j=3
b_{xj}	100	750	9000
b_{yj}	200	300	1000

Figure 4a. Parameters of database after local semi-joins

State	Attribute sizes					
	b_{x1}	b_{x2}	b_{x3}	b_{y1}	b_{y2}	b_{y3}
M_1	900	250	1000	800	700	9000
M_2	-	250	1000	720	700	9000
M_3	-	250	1000	-	700	9000

State	Attribute selectivities					
	p_{x1}	p_{x2}	p_{x3}	p_{y1}	p_{y2}	p_{y3}
M_1	0.9	0.25	0.1	0.8	0.7	0.9
M_2	-	0.25	0.1	0.72	0.7	0.9
M_3	-	0.25	0.1	-	0.7	0.9

State	Relation sizes			
	s_1	s_2	s_3	s_4
M_1	10000	20000	30000	40000
M_2	10000	20000	27000	40000
M_3	8000	20000	27000	4000

State	Common selectivities					
	COM_{xy1}	COM_{xy2}	COM_{xy3}	COM_{yx1}	COM_{yx2}	COM_{yx3}
M_1	1	1	1	1	1	1
M_2	-	1	1	0.9	1	1
M_3	-	1	1	-	1	1

State	Broadcast decisions
M_1	d_{x1} complement; cost:100; selectivity: 0.9
M_2	d_{y1} complement; cost:200; selectivity: 0.9
M_3	d_{x2} normal; cost:250; selectivity:0.25

Figure 4b. Database state in steps 1, 2, and 3 of heuristic scheduling

Table 1. The result of scheduling and estimated costs

	complement broadcasts										
Sequence	d_{x1}	d_{y1}	d_{x2}	d_{y2}	d_{x3}	d_{y3}	R_1	R_2	R_3	R_4	Total
Cost	100	200	250	175	700	225	5600	12600	675	100	20625

size, the first few broadcasts are very important and may dominate the overall transmission cost. A good heuristic should also enumerate the different broadcast order for the first few attributes. The order of broadcast for other attributes can be found by the greedy algorithm.

5. CONCLUSION

In this paper we have studied the distributed-database design on local multiaccess networks. Due to the broadcast capability of multiaccess networks, updates can be propagated easily, and information sent can be utilized by all the stations simultaneously. File allocation on these networks can be solved efficiently. The simple file-allocation problem is NP-hard on general networks. However, the problem is polynomially solvable here. In addition, by including an average-delay constraint, the problem can be transformed into the standard 0-1 knapsack problem which can be solved efficiently by dynamic programming or fully-polynomial-time approximation schemes. The general file-allocation problem with capacity constraints is still NP-hard. In solving the distributed query-processing problem, the semi-join concept with complement transmissions is adopted. A broadcast attribute can be utilized concurrently by many stations to reduce their sizes. This leads to less data transfer as compared to conventional strategies. A cost model for assessing the amount of data transmission is developed.

The problems on redundant materialization is not studied here. However, the problem of query scheduling is more complicated. First, when redundant information exists, it is necessary to identify the unique and minimal information to broadcast. This identification should be embedded in the scheduling process. Second, the reduction effect of a broadcast attribute on another attribute can no longer be assumed to be carried by the selectivity because the two attributes may overlap. In the extreme case, an attribute contains redundant information as the broadcast one, and there is no reduction. To solve this problem, it is found that the propagation of reduction effects actually starts from the local semi-joins. Assuming that all the attributes before local semi-joins are not overlapped, the sizes and selectivities can be estimated. As local semi-joins are carried out, the propagation of reduction effects are carried in the parameters COM. These parameters contain the common selectivities and should be used in global semi-join scheduling. The analysis of redundant materialization will be reported in a later paper.

REFERENCES

[APE83] Apers, P. M. G., A. R. Hevner and S. B. Yao, "Optimization Algorithms for Distributed Queries," *IEEE Trans. on Soft. Engr.*, Vol. SE-9, No. 1, Jan. 1983, pp. 57-68.

[BER81a] Bernstein, P. A., and D. M. W. Chiu, "Using Semi-Joins to Solve Relational Queries," *JACM*, Vol. 28, No. 1, Jan. 1981, pp. 25-40.

[BER81b] Bernstein, P. A., and N. Goodman, "Concurrency Control in Distributed Database Systems," *ACM Computing Surveys*, Vol. 13, No. 2, June 1981, pp. 185-221.

[CER83] Ceri, S., Navathe, S., and Wiederhold, G., "Distribution Design of Logical Database Schemas," *IEEE Trans. on Soft. Engr.*, Vol. SE-9, No. 4, July 1983, pp. 487-504.

[CHU69] Chu, W. W., "Multiple File Allocation in a Multiple Computer System," *IEEE Trans. on Comp.*, Vol. C-18, No. 10, Oct. 1969, pp. 885-889.

[CHU82] Chu, W. W., and Hurley, P., "Optimal Query Processing For Distributed Database Systems," *IEEE Trans. on Comp.*, Vol. C-31, No. 9, Sep. 1982, pp. 835-850.

[COD70] Codd, E. F., "A Relation Model of Data for Large Shared Data Banks," *CACM*, Vol. 13, 1970, pp. 377-387.

[DOW82] Dowdy, L. W., and D. V. Foster, "Comparative Models of the File Assignment Problem," *ACM Computing Surveys*, Vol. 14, No. 2, June 1982, pp. 287-313.

[ESW74] Eswaran, K. P., "Placement of Records in a File and File Allocation in a Computer Network," *Information Processing 74*, IFIPS, North Holland Publishing Co., 1974.

[GAR79] Garey, M. R., and D. S. Johnson, *Computers And Intractability: A Guide to the Theory of NP-completeness*, W. H. Freeman and Company, San Francisco, 1979.

[GEO72] Geoffrion, A. M., and R. E. Marsten, "Integer Programming: A Framework and State-of-the-Art Survey," *Management Science*, Vol. 18, No. 9, May 1972, pp. 465-491.

[GOU81] Gouda, M. G., and Dayal, U., "Optimal Semi-join Schedules for Query Processing in Local Distributed Database Systems," *Proc. ACM SIGMOD*, May 1981, pp. 164-175.

[HEV79] Hevner, A. R., and S. B. Yao, "Query Processing in Distributed Database Systems," *IEEE Trans. on Soft. Engr.*, Vol. SE-5, No. 3, May 1979, pp. 177-187.

[NEM69] Nemhauser, G., and Z. Ullman, "Discrete Dynamic Programming and Capital Allocation," *Management Science*, Vol. 15, No. 9, 1969, pp. 494-505.

[TAN81] Tanenbaum, A. S., *Computer Networks*, 1981, Prentice Hall, N. J..

[RAM83] Ramamoorthy, C. V., and B. W. Wah, "The Isomorphism of Simple File Allocation," *IEEE Trans. On Comp.*, Vol. C-32, No. 3, March 1983, pp. 221-232.

[WAH84] Wah, B. W., "File Placement on Distributed Computer Systems," *IEEE Computer*, Vol. 17, No. 1, Jan. 1984, pp. 23-32.

[WON77] Wong, E., "Retrieving Dispersed Data From SDD-1: A System for Distributed Databases," *Proc. 2nd Berkeley Workshop on Distributed Data Management and Computer Networks*, May, 1977, pp 217-235.

[YAO79] Yao, S. B., "Optimization of Query Evaluation Algorithms," *ACM Trans. On Database Systems*, Vol. 4, No. 2, June 1979, pp. 133-155.

[YU82] Yu, C. T., Chang, C. C., and Chang, Y. C., "Two Surprising Results in Processing Simple Queries in Distributed Databases," *Proc. COMSAC 82*, Nov. 82, pp. 377-384.