File Allocation in Distributed Databases with Interaction between Files

by

C.T. Yu, M.K. Siu, K. Lam and C.H. Chen

C.H. Chen and C.T. Yu: Department of Electrical Engineering and
Computer Science, University of Illinois at Chicago, Chicago, Illinois 60680
K. Lam: Department of Statistics, Hong Kong University, Hong Kong.
M.K. Siu: Department of Mathematics, Hong Kong University.

In this paper, we re-examine the file allocation problem. Because of changing technology, the assumptions we use here are different from those of previous researchers. Specifically, the *interaction of files during* processing of queries is explicitly incorperated into our model and the cost of communication between two sites is dominated by the amount of data transfer and is independent of the receiving and the sending sites. We study the complexity of the file allocation problem using the new model. Unfortunateiy, the problem is NP-hard. We present an approach to three versions of the problem, thus demonstrating the flexibility of our approach. We further argue that our method provides a practical solution to the problem, because accurate solutions are obtained, the time complexity of our algorithm is much smaller than existing algorithms, the algorithm is conceptually simple, easy to implement and is adaptive to users' changing access patterns.

Section 1. Introduction

In a distributed database system, files are assigned to various sites. A user query at a particular site may refer to data which is not available at that site. To answer the query, data is shipped across sites. An update of a particular file will cause the propagation of the update to all copies of that file in the network in order to maintain consistency of data. Thus, data communication is incurred for certain retrieval and update queries.

In this paper, the allocation of files in computer networks is investigated. The problem is defined as : given a network, a set of retrieval and update queries, determine the allocation of files such that the total communication cost is optimized.

Many models of the file allocation problem have been proposed [Case, Chan, ChLi, Chu, DoFo, FiHo, LamY, LoPo, MaRi, MoLe, UrOI, Wah, Whit]. However, previous formulations of the problem usually assume that files are independently accessed and therefore files are assigned indepentently. Current research in distributed query processing [HeYa, Chang, SDD1, YLCC,etc.] reveals that interactions between files during the processing of queries are common. In section 2, an example is given to illustrate that assignment of files to sites independently is unrealistic. This motivates us to incorporate the interaction between files directly into our model of the file allocation problem. Recently, an approach to incorporate interactions between files was given by [Aper]. However, his model is different from ours and the number of variables in his model grows rapidly to become unmanageable as the number of query types increases.

Previous models of file allocation assume that the cost of transferring data depends on the sending and the receiving sites. This assumption is crucial in showing that the file allocation problem is NP-hard [Eswa], implying that a polynomial solution is extremely unlikely. Current research indicates that the amount of data transfer is usually the dominant factor in communication cost and the cost is relatively independent of the sending and the receiving sites. This is true for satellite transmission and for many local networks. With this new assumption, we need to re-examine the complexity of the file allocation problem, which taking into consideration the interaction between files. Unfortunately, the problem is NP-hard, even for a star-network as given in section 3.1.2.

Since the interaction between files is rather complicated, we first study its effect on a star network with file partitioning. Then its effects on a star network with file clustering and finally on an arbitrary network are studied. In each case, we give a very simple algorithm and present the nice properties satisfied by the algorithms. The flexibility of our approach is clearly demonstrated by the similarities of the algorithms on the three versions of the file allocation problem. Experimental results on the first version of the file allocation problem reveals that our algorithm requires much less time as compared to other existing file allocation algorithms and yet obtain solutions which are on the average .27% away from the optimal solutions. In section 6, we present arguments that our algorithm is a practical one for the file allocation problem.

Section 2. An Example To Describe The Interaction Between Files

Suppose we have a single query accessing two relations [Ullm]. Let the query be {(R1.A,R2.C)|R1.B=R2.B} and the relations be R1(A,B) and R2(B,C), each of which can be implemented as a file. Let the contents of the relations be

A	B
a_1	0
.	.
.	.
.	.
a_{n-1}	0
a_n	2

R1

B	C
1	c_1
.	.
.	.
.	.
1	c_{m-1}
2	c_m

R2

Figure 2.1

(1) Suppose the assignment of R1 is independent from that of R2 and interaction between files during the processing of the query is ignored.

If the query originates from a site which is different from the site containing R1, then it is likely that the entire relation R1 has to be shipped to the query site. A similar situation applies to relation R2. Thus, data transferred in this case will be |R1| + |R2|, if both files are absent from the query site. As a result, the assignment of files R1 and R2 to the query site seems to imply a saving of |R1| + |R2| for the processing of the query over the non-assignment of the files to the query site.

(2) On the other hand, let us consider the interaction of files which is common in current distributed query processing strategies [Chang, HeYa, SDD1, YLCC, YCTBL].

(Case 1) R1 and R2 are placed at the same site which is different from the site the user query originates. Then the answer to the query is (a_n,c_m). Thus, the data transfer between the query site and the site containing R1 and R2 is (a_n,c_m). Another possibility is to send the parts of R1 and R2 that are needed to construct the answer to the query site. To illustrate this, let us temporarily assume the contents of R1 and R2 are

A	B
a_1	0
a_2	0
.	.
.	.
a_{n-1}	0
a_n	2
a_{n+1}	2
a_{n+2}	2

R1

B	C
1	c_1
1	c_2
.	.
.	.
1	c_{m-1}
2	c_m
2	c_{m+1}
2	c_{m+2}

R2

Then the answer of the query is

a_n	c_m
a_n	c_{n+1}
a_n	c_{m+2}
a_{n+1}	c_m
a_{n+1}	c_{m+1}
a_{n+1}	c_{m+2}
a_{n+2}	c_m
a_{n+2}	c_{m+1}
a_{n+2}	c_{m+2}

while the parts of R1 and R2 that are needed to construct the answer (denoted by R1' and R2' respectively) are

a_n	2
a_{n+1}	2
a_{n+2}	2

R1'

2	c_m
2	c_{m+1}
2	c_{m+2}

R2'

In this situation we can send R1' and R2' to the query site instead of the answer since it causes less amount of data transfer.

(Case 2) If R1 and R2 are at different sites, say s_1 and s_2 respectively, we can transfer (we refer back to Fig. 2.1) R1(B)={0,2} (or R2(B)={1,2}) to site s_2 (s_1). After the data arrive at site s_2 (s_1), take the join of R1(B) (R2(B)) with R2 (R1). (This is known as the semi-join [HeYo, SDD1, BeCh] from R1 (R2) to R2 (R1) on attribute B). R2 (R1) is then reduced to $(2,c_m)$ ($(a_n,2)$). Then R2(B)={2} (R1(B)={2}) is sent back to site s_1 (s_2) to reduce R1 (R2). Finally, the reduced R1 and the reduced R2 are sent to the query site to construct the answer.

In this example, the amount of data transfer when interaction of files is taken into consideration during the processing of the query is much smaller as compared to that when files are treated independently in both cases 1 and 2. As a result, placing files R1 and R2 at the query site gives a marginal saving. This example illustrates that the amount of data transfer in processing a query in a realistic environment can be drastically different from that in which files are independently accessed and independently assigned.

Section 3. File Allocation In The Star Network

We will first consider an important type of computer networks, the star network, where there is a single central computer which is connected to several regional computer sites, say m sites, [Chan, LamY, KeTY] (see Figure 3.1). The central computer site contains all files, say n files, and we want to determine the allocation of files in each regional computer site. We consider two versions of this problem. In the first version, the files are partitioned in regional sites i.e. the union of files in the regional sites is the set of all files and no two regional sites have a file in common. In this version, the regional computers may have limited storage capabilities and we may require that there is no redundant copies for each file in the regional computer sites. Therefore, we disregard update cost. In the second version, each file may have redundant copies and therefore update cost, as well as the retrieval queries cost, is taken into consideration.

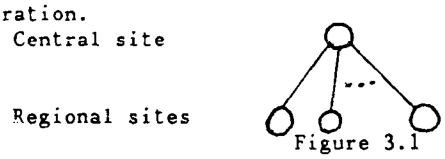

Figure 3.1

3.1. File Partitioning

In this subsection, we concentrate on the partitioning of files. Our aim is to find a partitioning of files in the regional sites such that the total communication cost for answering a set of retrieval queries is minimum.

3.1.1. Problem Formulation

Since the central site contains all files, a retrieval query accessing a set of files which are not available at the local site is routed to the central site. Then, the central site will send back either the answer or the parts of the accessed files which are needed to construct the answer, depending on which strategy requires less data transfer.

To simplify notation and for ease of understanding, we consider queries, each accessing one or two files, although the following results are true in general. Again for simplicity, the cost of routing queries to the central site is ignored; the communication cost is expressed in terms of the amount of data transfer. All these restrictions are not essential and can be eliminated by minor changes in the analysis.

Retrieval Query Cost

(Type 1): a query originating at site s requests for data contained in a single file, say file i,: ${}_{i}R_{s}$

If file i is available at site s, then the query can be answered without incurring any communication cost; otherwise, the answer with size F({i},R) is sent from central site to site s. Let y_{is} be a binary variable indicating the presence or absence of file i at site s, i.e. $y_{is} = 1$, if file i is at site s, 0, otherwise. Thus the cost for answering the query ${}_{i}R_{s}$, is

$$RC({}_{i}R_{s} | y_{is}=0) = F(\{i\},R) \quad \ldots\ldots\ldots\ldots (3.1)$$

$$RC({}_{i}R_{s} | y_{is}=1) = 0 \quad \ldots\ldots\ldots\ldots (3.2)$$

(Type 2): ${}_{ij}R_{s}$ i.e. a query originating at site s requests for the joint access of certain parts of files i and j.

Let F({i},R) and F({j},R) be the sizes of the parts of files i and j needed to construct the answer respectively (please refer to the example in section 2) and F({i,j},R) be the size of the answer. etc. are computed when the query is excuted). If both files are not at site s, then either the answer having size F({i,j},R) is transmitted to site s or the two parts having total size F({i},R) + F({j},R) are sent to site s, where the answer is constructed from the two parts, depending on which has a lower cost. Similarly, if only one file, say file i, is at site s, then either the answer or the part of file j with size F({j},R) is transmitted to site s. Thus, we have the cost function

$$RC({}_{ij}R_{s} | y_{is}=0, y_{js}=0) = \text{Min.}\{F(\{i,j\},R), F(\{i\},R)+F(\{j\},R)\} \quad (3.3)$$

$$RC({}_{ij}R_{s} | y_{is}=1, y_{js}=0) = \text{Min.}\{F(\{i,j\},R), F(\{j\},R)\} \quad (3.4)$$

$$RC({}_{ij}R_{s} | y_{is}=0, y_{js}=1) = \text{Min.}\{F(\{i,j\},R), F(\{i\},R)\} \quad (3.5)$$

$$RC({}_{ij}R_{s} | y_{is}=1, y_{js}=1) = 0 \quad (3.6)$$

In general, if a query originating at site s accesses a set of files T1 and if only a subset of files, say T2, are present at site s, then either the answer which is constructed from the set of files T1 at the central site is sent to site s or the parts of the files in T1 - T2, which are needed to answer the query are sent from the central site to site s. The strategy with the lower cost is chosen.

Objective Function:

Our objective is to find a partitioning of files to sites such that the total communication cost for answering a given set of queries is minimized. That is,

To minimize

$$\sum_{s}\{\sum_{i}\sum_{{}_iQ_s} RC({}_iR_s|y_{is})+\sum_{\{i,j\}}\sum_{{}_{ij}Q_s} RC({}_{ij}R_s|y_{is},y_{js})\} \qquad (3.7)$$

subject to

$y_{is} = 0, 1$ for $i=1,\dots,n$, $s=1,\dots,m$;

$\sum_{s} y_{is}=1$ for $i=1,\dots,n$. (There is one copy of each file at all the regional sites)

where ${}_iQ_s$ is the set of queries accessing file i and originating from site s, ${}_{ij}Q_s$ is the set of queries joint accessing files i and j and originating from site s. $\sum_{\{i,j\}}$ is summation over 2-element subsets of files $\{1,\dots,n\}$.

3.1.2. The Complexity of File Partitioning

If the cost of communicating between two sites is independent of the sending site and the receiving site, then the file allocation (e.g. [Case]) which does not take into consideration the interaction between files becomes a trivial problem. Under our new environment where communication cost is dominated by the amount of data transfer and the interactions between files are taken into consideration, the complexity of the file allocation problem is now re-examined.

We will show that the file partitioning problem (FPP) in the star network is NP-hard by reducing the exact cover by 3-sets (X3C) problem to the decision problem of FPP. The decision problem of the file partitioning problem (DFPP) is defined as follows.

[DFPP]: Given a star network with m regional sites $S=\{s_1,\dots,s_m\}$, n files $FE=\{f_1,\dots,f_n\}$,a set of queries QE and a cost parameter d, is there a partition of files to sites such that the total cost is less than or equal to d?

If a partition of the files to sites with the minimum cost can be found, then by comparing the minimum with d, we can clearly answer the DFPP. Thus, finding the minimum cost partition is at least as hard as DFPP. The exact cover by 3-sets problem was shown to be NP-complete in [GaJo] and is defined as follows.

[X3C]: Set X with 3q elements and a collection C of 3-element subsets of X with $|C| = m$. Does C contain an exact cover for X, i.e. a subcollection C' of C such that every element of X occurs in exactly one member of C'.

Proposition 3.1: The DFPP is NP-complete.

Proof: It is easy to see that DFPP is in NP since a nondeterministic algorithm can guess a partition of files and check in polynomial time whether that partition has cost ≤ d. We now show that X3C reduces to DFPP.

Given an instance, (X,C), of X3C, we construct an instance of DFPP defined as follows.

File Set FE: There is a file f_i corresponding to each element x_i in set X.

Site Set S: For each c_i in C there is a corresponding site s_i.

Query Set QE:

For each $c_i = (x_{i,1},x_{i,2},x_{i,3})$, $1 \leq i \leq m$, there are 3 retrieval queries each originating at site s_i and accessing two of the three corresponding files:

(1) ${}_{f_{i,1}f_{i,2}}R_{s_i}$ (2) ${}_{f_{i,2}f_{i,3}}R_{s_i}$ (3) ${}_{f_{i,3}f_{i,1}}R_{s_i}$

We also define the size of the answer $F(\{f_{i,j},f_{i,k}\},R)$ for each query to be 1 and $F(\{f_{i,j},f_{i,k}\},R) < \text{Min.} \{ F(\{f_{i,j}\},R), F(\{f_{i,k}\},R)\}$. By (3.3) to (3.6) the cost of answering each query is either 0, in the situation that both referred files are in the query site, or 1, otherwise.

Bound d:

The bound d is defined to be (m-q) * 3.

It is clear that the construction of DFPP can be carried out in polynomial time. We now show that DFPP has a partition with cost ≤ d iff (X,C) has an exact cover.

We first show that if (X,C) has an exact cover, then DFPP has a partition with cost ≤ d.

Let C' be an exact cover for (X,C). Assume without loss of generality that

$$c_{i_1}=(x_{i_1,1},x_{i_1,2},x_{i_1,3})\dots c_{i_q}=(x_{i_q,1},x_{i_q,2},x_{i_q,3})$$

are the elements of C'. We can assign files $f_{i_k,1}$, $f_{i_k,2}$, $f_{i_k,3}$ to site s_{i_k} for $1 \leq k \leq q$. The file assignment is a partition since C' is an exact cover. The total cost to answer the 3m queries is (m-q) * 3 since the 3q queries corresponding to the elements in C' in site s_{i_k}, $1 \leq k \leq q$, are answered with cost = 0 and each of the remaining (m-q) * 3 queries is answered with cost = 1.

We now show the reverse statement i.e. if there is a partition with cost ≤ d for DFPP, then there is an exact cover in (X,C). To show this statement, we will first show, in Lemma 3.2, that if there is a partition with cost ≤ d, the partition must have q sites each containing 3 files and the cost = d.

It is then easy to see that if a partition has q sites each containing 3 files and the cost is equal to d, then the queries originating from those q sites should be answered with cost 0.

Let the q sites be $s_{i_1},\dots,s_{i_q}$ and the queries be $f_{i_k,1}f_{i_k,2}{}^{R}s_{i_k}, f_{i_k,2}f_{i_k,3}{}^{R}s_{i_k}$ and $f_{i_k,3}f_{i_k,1}{}^{R}s_{i_k}$, $1 \le k \le q$ Then the corresponding 3-set elements $(x_{i_k,1}, x_{i_k,2}, x_{i_k,3})$, $1 \le k \le q$, form an exact cover for (X,C).

■

Lemma 3.2:

If there is a partition with cost ≤ d,the partition must have q sites each containing 3 files and the cost is equal to d.

Proof: Suppose the partition consists of m_i sites, each containing i files,$1 \le i \le h$, for some integer h such that $\sum_{i=1}^{h} m_i * i = 3q$.

We now compute the upper bound of the number of queries each of which is answered with cost = 0.

case 1: for those sites each containing a single file. No queries can be answered with cost = 0. since each query requests for information from 2 files.

case 2: for those sites each containing two files. At most one query in each site can be answered with cost = 0.

case 3: for those sites each containing more than two files. At most 3 queries in each site can be answered with cost = 0, because the number of queries submitted at each site is 3.

So, the upper bound of the number of queries which can be answered with cost = 0 is $m_2+3(m_3+\dots+m_h)$. Thus the cost of answering queries for this partition $\ge 3m-[m_2+3(m_3+\dots+m_h)]$. Since the partition has cost $\le (m-q) * 3$,

$m_2+3(m_3+\dots+m_h) \ge 3q$ ......(1)

However, $\sum_{i} m_i * i = 3q$ and $m_i \ge 0, i=1,\dots,h$, (2)

From (1) and (2), $m_i = 0$ for $i \ne 3$ and $m_3 = q$.

■

3.1.3. Necessary and Sufficient Conditions

In practice, users in a particular regional site, say s, may retrieve some files very often while they are retrieved rarely at other sites. Intuitively, those files should be placed at site s and not at any other site. To characterize this, a sufficient condition and a necessary condition for a file to be optimally assigned to a site will be derived.

Let J be a partition of files to sites such that file i is assigned to site t. Consider the reallocation of file i from site t to site s. The cost of answering a query ${}_{i}R_s$ in ${}_{i}Q_s$ (the set of queries originating at site s and accessing file i) will decrease from $RC({}_{i}R_s|y_{is}=0)$ to zero. For a query ${}_{ij}R_s$ in ${}_{ij}Q_s$, the amount of cost decrement is $RC({}_{ij}R_s|y_{is}=0,y_{js}) - RC({}_{ij}R_s|y_{is}=1,y_{js})$, which depends on the value of y_{js}. By Lemma 3.3, the minimun cost decrement is attained when $y_{js}=0$. Thus, there is a minimum gain of allocating file i to site s for all queries of the forms ${}_{ij}Q_s$ and ${}_{i}Q_s$. This minimum gain is

$$\sum_{{}_{i}Q_s} RC({}_{i}R_s|y_{is}=0)+\sum_{\substack{j\\ j\ne i}} \sum_{{}_{ij}Q_s} [RC({}_{ij}R_s|y_{is}=0,y_{js}=0) - RC({}_{ij}R_s|y_{is}=1,y_{js}=0)] \quad (3.8)$$

where $\sum_{{}_{i}Q_s}$ is summing over all queries originating from site s and accessing file i and $\sum_{\substack{j\\ j\ne i}} \sum_{{}_{ij}Q_s}$ is summing over all queries from the same site and accessing all pairs of files of the form (i,j) with different values of j.

On the other hand, the reallocating of file i from site t to site s will increase the cost of answering a query ${}_{i}R_t$ in ${}_{i}Q_t$ from 0 to $RC({}_{i}R_t|y_{it}=0)$. Similarly, the cost of answering a query ${}_{ij}R_t$ will increase by $RC({}_{ij}R_t|y_{it}=0,y_{jt}) - RC({}_{ij}R_t|y_{it}=1,y_{jt})$, which is also dependent on the value of y_{jt}. By Lemma 3.3, the maximum loss (cost increment) is attained when $y_{jt}=1$. Thus, the maximum loss of not allocating file i to site t, due to all queries of the form ${}_{ij}Q_t$ and ${}_{i}Q_t$ is

$$\sum_{{}_{i}Q_t} RC({}_{i}R_t|y_{it}=0) + \sum_{\substack{j\\ j\ne i}} \sum_{{}_{ij}Q_t} RC({}_{ij}R_t|y_{it}=0,y_{jt}=1)$$

The cost of answering queries which are not of the forms $\{ {}_{i}Q_s, {}_{i}Q_t, {}_{ij}Q_s, {}_{ij}Q_t \}$ will not change when file i is reallocated from site t to site s.

Proposition 3.4 says that if the minimum gain of allocating file i at site s is greater than the maximum loss of not allocating file i at site t, for every $t \ne s$, then every optimal solution consists of assigning file i to site s.

Lemma 3.3: For any pair of files i and j and any site s,

$$RC({}_{ij}R_s|y_{is}=0,y_{js}=0) - RC({}_{ij}R_s|y_{is}=1,y_{js}=0)$$
$$\le RC({}_{ij}R_s|y_{is}=0,y_{js}=1) - RC({}_{ij}R_s|y_{is}=1,y_{js}=1) \quad (3.10)$$

Proof: To prove (3.10) is equivalent to prove the following inequality

$$\text{Min. } \{F(\{i,j\},R),\ F(\{i\},R) + F(\{j\},R)\} - \text{Min. } \{F(\{i,j\},R),\ F(\{j\},R)\} \le \text{Min. } \{F(\{i,j\},R),\ F(\{i\},R)\} .$$

The inequality is true in each of the following cases :

(1) $F(\{i,j\},R) > F(\{i\},R) + F(\{i\},R)$.

(2) $\text{Max. } \{F(\{i\},R), F(\{j\},R)\} < F(\{i,j\},R) \le F(\{i\},R) + F(\{j\},R)$.

(3) $\text{Min. } \{F(\{i\},R), F(\{j\},R)\} < F(\{i,j\},R) \le \text{Max. } \{F(\{i\},R), F(\{j\},R)\}$.

(4) $F(\{i,j\},R) \le \text{Min. } \{F(\{i\},R), F(\{j\},R)\}$.

■

Proposition 3.4

If the following inequality is satisfied, then every optimal file partitioning should have file i assigned to site s.

Expression (3.8) > Expression (3.9), (3.11)

for any $t \neq s$.

Proof: Suppose the inequality (3.11) is satisfied and J is an optimal partition where file i is assigned to site t, $t \neq s$. Let J' be a new partition by reallocating file i from site t to site s. Let C(J) and C(J') be the cost of J and J' respectively.

$C(J') \leq C(J)$ - expression(3.8) + expression(3.9)

$< C(J)$ (by Lemma 3.3 and satisfying 3.11)

a contradiction of J being an optimal. ■

Similarly, a necessary condition for placing file i at site s is

Proposition 3.5

If file i is optimally placed at site s, then the following inequality is satisfied.

$$\sum_{{}_iQ_s} RC({}_iR_s \mid y_{is}=0) + \sum_{\substack{j \\ j\neq i}} \sum_{{}_{ij}Q_s} RC({}_{ij}R_s \mid y_{is}=0, y_{js}=1)$$

$$\geq \sum_{{}_iQ_t} RC({}_iR_t \mid y_{it}=0) +$$

$$\sum_{\substack{j \\ j\neq i}} \sum_{{}_{ij}Q_t} [RC({}_{ij}R_t \mid y_{it}=0, y_{it}=0) - RC({}_{ij}R_t \mid y_{it}=1, y_{jt}=0)]$$

for each $t \neq s$. (3.12)

The left hand side of the above inequality is the maximum gain of having the file at site s while the right hand side is the minimum loss of not having the file at site t. We will leave out the proof.

Note:

It is clear that the expression in (3.9) is also the maximum gain of allocating file i to site t. Thus, later in this section, we will use the terms gain and loss interchangeably.

In general, a query may access more than two files. To derive the necessary condition and the sufficient condition for the general situation, it is sufficient to derive the minimun gain and maximum gain of allocating file i at site s for each query which access a set of files, say T.

Let $T = \{i, j_1, \ldots, j_k\}$ and ${}_TR_s$ denote the query accessing the set of files T and originating at site s. It is easy to show that the minimum gain of allocating file i to site s for query ${}_TR_s$ is

$RC({}_TR_s \mid y_{is}=0, y_{j_1s}=0, \ldots, y_{j_ks}=0)$

$- RC({}_TR_s \mid y_{is}=1, y_{j_1s}=0, \ldots, y_{j_ks}=0)$

and the maximum gain is $RC({}_TR_s \mid y_{is}=0, y_{j_1s}=1, \ldots, y_{j_ks}=1)$.

Thus, Proposition 3.4 and 3.5 are generalizable to queries accessing any number of files.

It is clear that (3.11) can be used to allocate certain files to certain sites optimally and (3.12) can be used to prevent the allocation of some suboptimal allocation of files to sites. However, the checking of the satisfaction of (3.11) and (3.12) can be expensive. In the next subsection, we shall present an algorithm which will assign the files to the sites consistent to Proposition 3.4 and 3.5, without checking the satisfaction of (3.11) and (3.12).

3.1.4: An adaptive algorithm

The algorithm to be described consists of having two n by m matrix M and M1. M[i,s] is the "expected gain" of placing file i at site s. The algorithm has two phases. In the first phase, when no user access pattern is available, certain entries in the M matrix will be updated for each submitted user query to give the current expected gain. After a set of queries have been processed, if M[i,s] is the maximum entry in row i, then file i will be tentatively assigned to site s. In the second phase, the same process as in phase 1 is carried out except that the gain of placing file i at site s is computed based on the tentative assignments of files to sites obtained in phase 1. The gains, due to all processed queries, are stored in the matrix M1. If the file assignment as given by M is the same as that given by M1, then the algorithm terminates, otherwise, the gains of placing the files at the sites are re-computed, based on the file assignment given in M1 and the new gain is stored in M. This process is repeated until M and M1 give identical file assignments. In practice, each of the matrices M and M1 can be partitioned into m n-vectors such that each site stores one vector to keep track of the current gain. After a time period, those vectors are sent to the central site to determine the file allocation.

The details are given as follows:

Phase 1

```
step 1:
   Initialize M[i,s] and M1[i,s] to be 0, $1 \leq i \leq n$; $1 \leq s \leq m$.
step 2:
   For each query processed;
   case ( type of query )
(i) /* the query is ${}_iR_s$ */
   increase M[i,s] by $RC({}_iR_s \mid y_{is}=0)$.
(ii) /* the query is ${}_{ij}R_s$ */
   increase M[i,s] by
[$RC({}_{ij}R_s \mid y_{is}=0, y_{js}=0) - RC({}_{ij}R_s \mid y_{is}=1, y_{js}=0)$]*$w_j$
   + $RC({}_{ij}R_s \mid y_{is}=0, y_{js}=1)$ * $(1 - w_j)$.
/* Any $w_j$ lying between 0 and 1 can be used.
The above quantity is a weighted average of the
minimum and the maximum gains of placing file i
at site s. $w_j$ can be interpreted as the
probability that file j is not assigned to site
s. For simplicity, we assign $w_j$ to be 1/2, $1 \leq$
$j \leq n$. */
```

increase M[j,s] by

$[RC({}_{ij}R_s | y_{is}=0, y_{js}=0) - RC({}_{ij}R_s | y_{is}=0, y_{js}=1)] * w_i + RC({}_{ij}R_s | y_{is}=1, y_{js}=0) * (1 - w_i)$.

Step 3:

After many queries have been processed, if M[i,s] is the maximum entry in row i, then assign file i to site s.

Phase 2

We invoke step 2 and step 3, except that $w_j = 1$ if y_{js} has been determined to be 0, 0 otherwise for each $1 \le j \le n$, $1 \le s \le m$. The expected gain information is contained in matrix M1. If the file assignment given by M is not the same as that give by M1, then the iteration continues with M and M1 interchanged else the algorithm terminates.

Note: The above algorithm assumes that the query access pattern is unchanged from one time period to another. In practice, access pattern changes with respect to time and we proceed as follows. If the network is being set up, then we will invoke phase 1 and then terminates the algorithm as soon as a file allocation is determined in phase 2. No further iteration is performed. If the network has been operational for some time, then the current file assignments will determine the values of w_j's in (ii) for step 2. It is sufficient to invoke phase 2 to obtain the first file allocation and terminates the algorithm for the current time period.

We now give the time complexity of executing one phase of the algorithm.

Let q be the number of queries processed and k be the average number of files accessed per query in step 2. The time complexity of the algorithm is O(kq) for step 2 plus O(mn) for step 3. Since all other file allocation algorithm do not take into consideration the time to collect statistics, which is essentially step 2, the time complexity of this algorithm in relation to other file allocation algorithms, is O(mn) only. Clearly, this is optimal in time.

We expect the matrices M and M1 to contain mostly zero entries. For implementation, it is sufficient to store the non-zero entries.

It can be shown that the algorithm has the properties that (a) if the sufficient condition of allocating file i to site s is satisfied, then the algorithm will allocate file i to site s without checking the truth or falsity of inequality (3.11); (b) if the necessary condition to optimally allocate file i at site s is violated, then the algorithm will not allocate file i to site s without checking the inequality (3.12); and (c) Assume that users' access pattern is unchanged from one time interval to another. If an optimal solution is obtained at the k-th iteration in phase 2, then the optimal allocation remains unchanged at the (k+1)-th iteration. Furthermore, if the file allocation at the (k+1)-th iteration is different from that of k-th iteration, a strictly better solution than that given in the k-th iteration can be extracted from the solution at the (k+1)-th iteration. We now give the proof of property (a) but leave out the proofs of properties (b) and (c).

Proposition 3.6

If the sufficient condition of allocating file i to site s is satisfied, then the algorithm will allocate file i at site s.

Proof: It can easily be seen that the entry M[i,s] is the expected gain of allocating file i to site s by examining step 2 of the algorithm. So, the value of M[i,s] is less than or equal to the maximum gain of allocating file i at site s and greater than or equal to the expression of (3.8) (the minimum gain). By the same reasoning, M[i,t] is less than or equal to expression (3.9). If the sufficient condition is satisfied, then M[i,s] > M[i,t] for all $t \ne s$ and file i is allocated at site s by step 3 of the algorithm.

■

3.2: File Clustering

Problem Formation

The problem is the same as that of 3.1, except that a file may be assigned to zero or more regional site. As a result, update cost can be significant and is taken into consideration. We first consider the effect of the various types of queries on the communication cost between the central site and a particular regional site, say site s.

Retrieval Query Cost

(1) same as query type (1) in 3.1.

(2) same as query type (2) in 3.1.

Update Query Cost

(3) a query originating at site s, requests for the updating of a single file, say file i: ${}_iU_s$.

An update query of size $UM({}_iU_s)$ is sent to the central site, irrespective of the presence or absence of file i at site s. Then granting or rejecting messages are exchanged between the central site and the sites containing file i. The number of messages exchanged is a constant times the number of copies of the files [U11m]. This is refered to as the concurrency control cost. Thus, for each site containing a copy of the file, the communication cost incurred between the central site and the regional site is a constant. Let $CC({}_iU_s)$ = the constant of the concurrency control cost. Thus, the cost incurred between site s and the central site to answer the query are:

$$UC_s({}_iU_s | y_{is}=1) = UM({}_iU_s) + CC({}_iU_s) \quad (3.13)$$

$$UC_s({}_iU_s | y_{is}=0) = UM({}_iU_s) \quad (3.14)$$

(4) ${}_iU_t$: same as (3) except the query originating at site $t \ne s$. With similar reasoning as in (3), the cost incurred between site s and the central site are:

$$UC_s({}_iU_t | y_{is}=1) = UM({}_iU_t) + CC({}_iU_t) \quad (3.15)$$

$$UC_s({}_iU_t | y_{is}=0) = 0 \quad (3.16)$$

Suppose some copies of certain files are added to regional site t ≠ s. Any retrieval query originating at site s can be answered with the same cost, irrespective of the additional files. at site t. Furthermore, any retrieval query originating from any site other than s does not incur any transmission cost between site s and the central site. For update queries, the extra copies of the files at site t causes additional update cost. However, the additional cost occurs between the central site and the regional site t only. Thus, the transmission cost between site s and the central site remains unchanged, when additional copies of some files are added to regional site t. In other words, the allocation of files to each regional site is independent of other regional sites. So, our objective of minimizing total communication cost between the central site and the regional sites can be met by minimizing the communication cost between the central site and each regional site independently. Therefore, the subscript s of UC_s in (3.13) - (3.16) can be ignored. The objective of allocating a set of n files to a particular site s is to minimize

$$\sum_{i}\sum_{{}_iQ_s} RC({}_iR_s|y_{is}) + \sum_{\{i,j\}}\sum_{{}_{ij}Q_s} RC({}_{ij}R_s|y_{is},y_{js})$$

$$+\sum_{i}\sum_{{}_iP_s} UC({}_iU_s|y_{is}) + \sum_{i}\sum_{t\neq s}\sum_{{}_iP_t} UC({}_iU_t|y_{is}) \quad (3.17)$$

where ${}_iP_u$ is the set of update queries which update file i and originating at site u.

Necessary and Sufficient Conditions

Consider allocating file i to site s. A gain will be achieved for retrieval queries originating at site s and accessing file i (refer to section 3.1). However, the allocation of file i at site s will incur additional update cost. It can be easily seen if the minimum gain of allocating file i for retrieval queries to site s is greater than the additional update cost incurred, the file i should be assigned to site s. On the other hand, if the maximum gain of allocating file i at site s for retrieval query is less than the update cost incurred, then file i should not be assigned to site s. So, the sufficient condition and the necessary condition of allocating file i to site s can be easily derived. They are stated in Propositions 3.7 and 3.8.

Proposition 3.7

If file i at site s satisfies the following inequality, then every optimal file allocation should have file i assigned to site s.

$$\sum_{{}_iQ_s} RC({}_iR_s|y_{is}=0) +$$

$$\sum_{\substack{j\\ j\neq i}}\sum_{{}_{ij}Q_s}[RC({}_{ij}R_s|y_{is}=0,y_{js}=0)-RC({}_{ij}R_s|y_{is}=1,y_{js}=0)]$$

$$> \sum_{{}_iP_s} [UC({}_iU_s|y_{is}=1) - UC({}_iU_s|y_{is}=0)] +$$

$$\sum_{t\neq s}\sum_{{}_iP_t} UC({}_iU_t|y_{is}=1) \quad (3.18)$$

Proposition 3.8

If file i is optimally allocated at site s, then the following inequality is satisfied.

$$\sum_{{}_iQ_s} RC({}_iR_s|y_{is}=0) + \sum_{\substack{j\\ j\neq i}}\sum_{{}_{ij}Q_s} RC({}_{ij}R_s|y_{is}=0,y_{js}=1)$$

$$\geq \sum_{{}_iP_s} [UC({}_iU_s|y_{is}=1) - UC({}_iU_s|y_{is}=0)]$$

$$+ \sum_{t\neq s}\sum_{{}_iP_t} UC({}_iU_t|y_{is}=1) \quad (3.19)$$

Adaptive Algorithm

The adaptive algorithm described in section 3.1 can be used with a minor modification. In this case, an array B is stored such that B[i] is the expected gain of assigning file i to site s (refer to section 3.1). Also, another array C is stored such that C[i] is the update cost of assigning file i to site s. If B[i] > C[i] after a time period, file i will be assigned to site s, otherwise, file i is not assigned. The algorithm also has the same properties (a), (b) and (c) as stated in section 3.1.

Section 4. General Network

The network considered in this section is assumed to be an arbitrary point-to-point network. The query processing model is more complicated than that given in the previous section. In the star network, the central site contains a copy of each file. For a retrieval query originating at site s, the processing of the query is independent of the allocation of files in any other regional site, because the required files are either at the central site or at site s. In an arbitrary network, we may not have a site containing all files. A typical strategy[Chang, HeYa, KeTY, SDD1, YLCC, YCTBL] is to perform a sequence of semi-joins and then send the reduced relations to an assembly site to construct the answer. The common assumption in current research of distributed query processing is that the transmission cost between any two sites is not dependent on the sending and the receiving sites and the cost is a linear function of the amount of data transfered. This is valid for satellite communication and many local networks. For other networks, the communication cost is usually dominated by the amount of data transfer [Date2]. Here we make the same assumption. Again, the query types described in section 3 are used in this section. The cost model for the queries are as follows.

Retrieval Query Cost

(1) ${}_{i}R_{s}$: The cost is the same as that in 3.1.

(2) ${}_{ij}R_{s}$:

case 1: there is a site containing both files i and j.

If s is such a site, then the cost of answering the query is 0, otherwise the site containing both files can be visualized as the central site in the star network. Thus, equations (3.3)-(3.6) apply. Let x_{ijs} be a binary variable such that it is 1 if there is a site containing both files i and j and 0 otherwise. Let the cost function be denoted by $RC({}_{ij}R_{s}|y_{is},y_{js},x_{ijs})$. Then the following equation can easily be seen to be equivalent to the set of equations (3.3)-(3.6). For example, (3.3) is obtained by setting y_{is} = y_{js} = 0 in the equation.

$$RC({}_{ij}R_{s}|y_{is},y_{js},x_{ijs}=1) = \text{Min.}\{F(\{i,j\},R),(1-y_{is})*F(\{i\},R)+(1-y_{js})*F(\{j\},R)\} \quad (4.1)$$

case 2: there is no such site i.e. x_{ijs}=0.

A possible strategy which is common in current distributed query processing algorithm [Chang, HeYa, KeTY, SDD1, YLCC, YCTBL] is to execute a sequence of semi-joins (see example in section 2) to reduce files i and j to sizes F({i},R) and F({j},R) respectively. Then, each of the reduced files is sent to site s, if it is not already there. Let $SC({}_{ij}R_{s}|y_{is},y_{js})$ be the communication cost of the sequence of semi-joins executed. Then, the total cost is

$$RC({}_{ij}R_{s}|y_{is},y_{js},x_{ijs}=0) = SC({}_{ij}R_{s}|y_{is},y_{js})+(1-y_{is})*F(\{i\},R)+(1-y_{js})*F(\{j\},R) \quad (4.2)$$

Update Query Cost

Let the query which originates from site s and requests to update file i be denoted by ${}_{i}U_{s}$. Let n_i be the number of sites, excluding site s, which contain file i. One possible concurrency control scheme which guarantees consistency of data is as follows. A message is sent to each of the n_i sites requesting a lock. Assuming that the required locks are granted, the n_i sites will reply affirmatively. Then the update query is sent to all other sites containing copies of the file. Clearly, the total number of messages sent will be proportional to n_i. Thus, each of the n_i sites will be assigned a constant cost. This constant has two components: one component being the message cost due to the request for locking which is referred to as concurrency control cost and denoted by $CC({}_{i}U_{s})$ and the other being the propagation of the update query which is referred to as update cost and denoted by $UM({}_{i}U_{s})$. Thus, the total communication cost incurred by update query ${}_{i}U_{s}$ is

$$UC({}_{i}U_{s}) = n_i * [UM({}_{i}U_{s}) + CC({}_{i}U_{s})]$$

This is equivalent to assigning

$$UC({}_{i}U_{s}|y_{it}) = \begin{cases} 0 & t = s \\ y_{it}*[UM({}_{i}U_{s})+CC({}_{i}U_{s})] & t \neq s \end{cases} \quad (4.3)$$

Objective Function

Our objective is to find a file allocation such that the total communication cost is minimized subject to at least one copy of each file must be allocated in the network. That is, To minimize

$$\sum_{s} \sum_{i} \sum_{{}_{i}P_{s}} \sum_{t\neq s} UC({}_{i}U_{s}|y_{it})$$

$$+ \sum_{s} \sum_{i} \sum_{{}_{i}Q_{s}} RC({}_{i}R_{s}|y_{is})$$

$$+ \sum_{s} \sum_{\{i,j\}} \sum_{{}_{ij}Q_{s}} RC({}_{ij}R_{s}|y_{is},y_{js})$$

subject to

$$\sum_{s} y_{is} \geq 1 \quad \text{for } i = 1,\dots,n.$$

Necessary and Sufficient Conditions

The following properties can be easily proved by using the same ideas developed in previous section.

Proposition 4.1

If file i at site s satisfies the following condition, then every optimal solution should include file i at site s.

$$\sum_{{}_{i}Q_{s}} RC({}_{i}R_{s}|y_{is}=0) +$$

$$\sum_{\substack{j \\ j\neq i}} \sum_{{}_{ij}Q_{s}} [RC({}_{ij}R_{s}|y_{is}=0,y_{js}=0,x_{ijs}=1) - RC({}_{ij}R_{s}|y_{is}=1,y_{js}=0,x_{ijs}=1)]$$

$$> \sum_{t\neq s} \sum_{{}_{i}P_{t}} UC({}_{i}U_{t}|y_{is}=1) \quad (4.4)$$

Proposition 4.2

If file i is optimally allocated at site s, then the following inequality is satisfied.

$$\sum_{{}_{i}Q_{s}} RC({}_{i}R_{s}|y_{is}=0)+ \sum_{\substack{j \\ j\neq i}} \sum_{{}_{ij}Q_{s}} RC({}_{ij}R_{s}|y_{is}=0,y_{js}=1, x_{ijs}=0)$$

$$+ \sum_{t\neq s} \sum_{\substack{j \\ j\neq i}} \sum_{{}_{ij}Q_{t}} [RC({}_{ij}R_{t}|y_{it}=0,y_{jt}=0,x_{ijs}=0) - RC({}_{ij}R_{t}|y_{it}=0,y_{jt}=0,x_{ijs}=1)]$$

$$\geq \sum_{t\neq s} \sum_{{}_{i}P_{t}} UC({}_{i}U_{t}|y_{is}=1) \quad (4.5)$$

Algorithm

The adaptive algorithm applied to this enviroment is similar to those given in previous section except that a n by m matrix B is stored such that B[i,s] is the expected gain of allocating file i at site s, and another n by m matrix C is stored such that C[i,s] is the update cost of assigning file i at site s. When a query is submitted, some entries either in the B matrix or the C matrix will be updated using (3.1), (4.1), (4.2) and (4.3). Finally, assign file i to site s if B[i,s] > C[i,s]. In order to satisfy the constraint that at least one copy of each file has to be allocated, file i is assigned to the site with maximum B[i,s] - C[i,s] even if B[i,s] is less than C[i,s] for every s=1,...,m. The same properties (a), (b), and (c) stated in section 3.1 are also satisfied.

Section 5: Experimental Results

In this section, experimental results for the adaptive file partitioning algorithm in a star network are presented. Various problem sizes ranging from 2 sites with 8 files to 50 sites with 100 files, are used to test the speed and the accuracy of the algorithm. In order that the simulation is realistic, the access probabilities of the files at each site form an approximate Zipfian distribution with a randomly generated parameter [see Knut]. Furthermore, the volumes of traffic directed at the sites are different but vary by no more than a factor of 25. For each of the first 12 problem sizes listed in Table 1, 100 cases are run and the average is taken. For problem size 13, 50 cases and for problem sizes 14 and 15, 10 cases each are run and the average is taken. In each case, up to 6000 query types are generated by the computer where two queries are of different query types if they either access different sets of files or require different amounts of data. Each query accesses either one or two files. The weights w's for all files are arbitrarily set to 1/2 for phase 1 of the algorithm. The amounts of data requested by each query (i.e. F({i,j}, R), F({i}, R), etc.) are randomly generated. (The reason why we do not perform 100 cases for each of the larger problem sizes is that enormous amount of time has to be spent on verifying the numerous queries generated in fact satisfy the access patterns described above).

The speed of the algorithm is tested as follows. A batch of queries are processed in phase 1 of the algorithm to produce a file allocation. Then, another batch of queries (randomly drawn from the same query pattern) are further processed in phase 2 of the algorithm to produce another file allocation. If the two file allocations are identical then the algorithm terminates and the file allocation is obtained; otherwise another batch of queries are further processed. This will be continued until the file allocation produced by all queries up to the i-th batch is the same as the file allocation produced by all queries up to the (i-1)th batch, for the smallest positive integer i. A PDP 11-45 computer is used. It is seen from Table 1 that the average time used for the smallest problem size of 2 sites with 8 files is 0.69 sec. and the average time for the largest problem size of 50 sites with 100 files is 10.68 sec. Clearly, the execution time compares favorably with recent results reported by [FiHo] (the average execution time for 50 sites with 1 file is more than 1 min. by a DEC-10 computer) and by [ChLi] (the average execution time for 3 sites with 100 files is 28.62 min. by a PDP 11-70 computer), although the problem formulations are different and the algorithm by [ChLi] yields an optimal solution. (If their algorithms are run on the problem with 50 sites and 100 files, then their running times are at least more than an hour, which is a few hundred times more than ours).

We now proceed to discuss the accuracy of the algorithm. The accuracy of our algorithm is reported for the first 12 problem sizes only, because a branch-and-bound algorithm, which we use to obtain the optimal solution, does not converge for the larger problem sizes. (In fact, among the first 12 problem sizes, the branch-and-bound algorithm fails to terminate in some cases.) It is seen from table 1 that the algorithm usually has a higher chance of obtaining the optimal solution for smaller problem sizes (86 percent for the 2 sites with 8 files case) than for larger problem sizes (70 percent for the 4 sites with 9 files case), though the percentage of obtaining the optimal solution is not always monotonically decreasing with respect to increasing problem size. However, the average percentage difference between the solution generated by our algorithm and the optimal solution seems to decrease from the maximum of 0.40 percent for the 3 sites with 6 files case to the minimum of 0.126 percent for the 5 sites with 9 files case. The average error is 0.265 percent for all cases. The good performance exhibited by our algorithm can be explained by the fact that our algorithm will obtain the optimal solution if the sufficient condition is satisfied and will not assign a file to a site suboptimally if the necessary condition is violated. In other words, file having obvious optimal solutions (satisfying the sufficient condition) are optimally allocated by our algorithm, while files having obvious sub-optimal solutions (violating the necessary condition) are disallowed. Furthermore, properties (c) will not permit the algorithm to stop at a point in which a better solution can be easily obtained.

Section 6: Conclusion

We have described an adaptive file allocation algorithm. This algorithm is particularly suitable to a dynamic data base environment where users' access pattern may change from one time interval to another. The algorithm is invoked at the beginning of each time period. If there is a change to the users' access pattern, then the file assignment obtained by the algorithm in the current time period will be different from that obtained by the same algorithm in the previous time period. As a result, physical assignment of files to sites will take place. This can be considered as part of an adaptive database design [HaCh, HaNi, Salt, YSLT]. However, this is a very significant difference between our approach and earlier studies. In [HaCh, HaNi], all user queries are stored, classified into query types and the cost of executing each type of queries is estimated. Usually, the classification process leads to significant inaccurates. For example, different queries referring to same set of attributes (though having different conditions on the attributes) of a relation are treated to be of the same type and have the same processing cost. In spite of this crude classification, the number of query types is exponential, which makes the classification process expensive in space, because a frequency count has to be kept for each query type. More importantly, the estimation of the cost of processing a giving query type depends on the assumption that data is uniformly and independently distributed. (In figure 2.1, the applicable assumptions are "the values 0 and 2 occur with equal probabilities in R1(B); the distribution of the value {a1,...,an} and {0,2} in R1 are independent;" similar situations apply to R2). Clearly, such assumptions lead to substantial inaccuracies. On the other hand, our method consists of obtaining actual data requirement at the time a query is processed. The query data requirement is immediately incorporated into the gains or losses and the query need not be stored nor classified. Thus, the suboptimal solutions that our algorithm produces (.27% away from the optimal solutions in one version of the file allocation problem) are far more accurate than the solutions of other algorithms, which do not obtain accurate data transfer statistics for each query and/or do not consider the interaction between files.

We believe the approach presented here is a practical solution to the file allocation problem, because
(1) it gives accurate solutions;
(2) it is conceptually simple;
(3) it is adaptive to users' changing environment;
(4) its time complexity is much smaller than existing algorithms;
(5) it is very flexible as demonstrated by our applying the approach to three different versions of the problem.

REFERENCES

[Aper] Apers, P.M.G., "Redundant allocation of relations in a communication network" Proc. 5th Berkeley Workshop on Distributed Data Management and Computer Networks, pp.245-258, Feb. 1981.

[BeCh] Bernstein, P.A., Chiu, D.W. "Using semijoins to solve relational queries.", J. ACM. Jan. 1981, pp. 25-40.

[Case] Casey, R.G., "Allocation of copies of a file in an information network", AFIP Conference Proceeding, Vol. 40, 1972, Spring Joint Computer Conference, May 1972, pp. 612-625.

[Chan] Chang, S. K., "Database decomposition in a hierarchical computer system", ACM SIGMOD, May 1975, pp. 48-52.

[Chang] Chang, J. M., "A heuristic approach to distributed query processing", VLDB, 1982.

[ChLi] Chang, S. K. and Liu, A.C., "A database file allocation problem", IEEE COMPASC, 1981.

[Chu] Chu, W.W., "Optimal file allocation in a multi-computer information systems", IEEE Transaction on Computer, Vol. 18, No. 10, October 1969, pp. 885-889.

[Codd] Codd, E.F., "A relational model of data for large shared data banks", C.ACM, Vol. 13, No. 6, 1970, pp. 377-387.

[Date] Date, C.J., "An introduction to database systems", Addison-Wesley, Reading, Mass., 1977.

[Date2] Date, C.J., "An introduction to database systems", Addison-Weskey, Vol.2, Reading, Mass., 1983.

[DoFo] Dowdy, L.W. and Foster, D.V. "Comparative Models of the file assignment program", ACM Computing Survey, Vol. 14, No. 2, June, 1982, pp. 287-313.

[Eswa] Eswaran, K.P., "Placement of records in a file and file allocation in a computer network", IFIP, August 1974, pp. 304-307.

[FiHo] Fisher, M. and Hochbaum, D., "Database location in computer networks", J.ACM., 1980, pp. 718-735.

[GaJo] Garey, M.R., Johnson, D.S., "Computer and intracrability: A guide to the theory of NP-completeness", Freeman 1979.

[Ghos] Ghosh, S.P., "Distributing a data base with logical associations on a computer network for parallel searching", IBM Research Report, 1974.

[Grbe] Grapa, E. and Belford, G., "Some theorems to aid in solving the file allocation problem", CACM, November 1977, pp. 878-882.

[HaCh] Hammer, M. and Chan, A., "Index selection in a self-adaptive database management system", ACM SIGMOD, 1976, pp. 1-8.

[HaNi] Hammer, M. and Niamir, B., "A Heuristic approach to attribute paartitioning", ACM SIGMO, 1979, pp. 93-101.

[HeYa] Hevner, A. and Yao, S.B., "Query processing in distributed database systems", IEEE Trans on Software Engineering, Vol.SE-5, No. 3, May, 1979, pp. 177-187.

[HoSe] Hoffer, J.A. and Severance, D., "The use of cluster analysis in physical data base design", VLDB, 1975, pp. 69-86.

[KeTY] Kerschberg, L., Ting, P.D. and Yao, S.B.,"Query Optimization in star computer network", ACM TODS, Vol. 7, No. 4, December, 1982, pp. 678-711.

[LamY] Lam, K. and Yu, C.T., "An approximation algorithm for a file allocation problem in a hierarchical distributed system", ACM SIGMOD, 1980, pp. 125-132.

[LoPo] Loomis, M.S. and Popek, G.J., "A model for database destribution", Trends and Applications, Computer Methods, IEEE Computer Society, 1976.

[MaRi] Mahmoud, S. and Riordon, J.S., "Optimal allocation of Resources in distributed information networkd", ACM TODS, Vol. 1, No. 1, March 1976, pp. 66-78.

[MoLe] Morgan, H.L. and Levin, K.D., "Optimal program and data location in computer networks", CACM, 1977, pp. 315-322.

[Rive] Rivest, R., "On self-organizing sequential search heuristics", C. ACM, 1976, pp. 63-67.

[Salt] Salton, G., "Dynamic information and library processing", Prentice-Hall, Englewood Cliffs, New York, 1975.

[SDD1] Bernstein, P.A., Goodman, N., Wong, E., Reeve, C. and Rothnie, J.B., "Query processing in a system for distributed databases (SDD-1)", ACM TODS, Vol. 6, No. 4, Dec. 1981, pp. 602-605.

[Thom] Thomas, R.H., "A majority consensus approach to concurrency control", ACM Trans. on Database Systems Vol4, No2, 1979.

[Ullm] Ullman, J.D., "Principles of Database Systems", Computer Science Press, 1982.

[UrOI] Urano, y., Ono, K. and Inone, S., "Optimal design of distributed networks", Proc. of International Conference on Computer Communication, August 1974, pp. 413-420.

[Wah] Wah, B.W., "An efficient heuristic for file placement on distributed database", Proc. COMPSAC '80, IEEE COMPSAC, pp. 462-468.

[Whit] Whitney, V.K.M., "A study of optimal file assignment and communication network conformation in remote-access computer message processing and communication systems", Univ. of Michigan, Dept. of Electrical Engineering, 1970.

[YaDT] Yao, S.B., Das, K.S. and Teory, T.J., "A dynamic database reorganization algorithm", ACM TODS, pp.159/174

[YSLT] Yu, C.T., Siu,M.K., Lam, K. and Tai, F., "Adaptive clustering schemes: general framework", IEEE COMPSAC, Nov. 1981

[YuOz] Yu, C.T., Ozsoyoglu, M.Z., "An algorithm for tree-query membership of a distributed query". IEEE COMPSAC 1979, pp.306-312.

[YCTBL] Yu, C.T., Chang, C.C., Templeton, M., Brill, D., Lund, E. , "On the design of a query processing strategy in a distributed database environment", ACM SIGMOD 1983, pp.30-39.

[YLCC] Yu, C.T., Lam. K., Chang, C.C., Chang, S.K., "A promising approach to distributed query processing", Berkeley Conference on Distributed Data Base, Feb., 1982, pp.363-390.

problem size (no. of sites, no.of files)	average C.P.U. time in sec. PDP-11/45	average percentage from optimal	percentage of cases where optimal solutions are obtained
1. (2,8)	.69	0.21	86/100
2. (3,6)	.70	0.40	78/100
3. (3,7)	.79	0.279	83/100
4. (3,8)	.72	0.389	78/100
5. (3,9)	.88	0.303	74/100
6. (3,10)	.93	0.29	73/100
7. (4,8)	.87	0.297	70/99
8. (4,9)	.82	0.298	70/100
9. (4,10)	.90	0.198	78/100
10. (5,8)	.89	0.243	72/100
11. (5,9)	.89	0.126	73/99
12. (5,10)	1.01	0.143	71/97
13. (10,40)	2.78	----	----
14. (40,100)	9.75	----	----
15. (50,100)	10.68	----	----

Table 1