ADAPTIVE INFORMATION SYSTEM DESIGN

ONE QUERY AT A TIME*

by

C T Yu
Dept of EECS, University
of Illinois at Chicago

and

C H Chen
Bell Communication
Research

Abstract

We propose a new way to perform adaptive information system design An abstract information structure provides information about how an appropriate actual information structure should be Whenever a query is processed, the abstract information structure is modified to reflect the effect of the processed query After many queries are processed, the abstract structure is examined If it reveals that the actual information structure is inappropriate, then the actual structure will be modified, otherwise no change is made

This one-query-at-a-time approach is applied to two problems, namely the record clustering problem and the file allocation to distributed database problem Results demonstrate that our approach yields good performance, is conceptually very simple, is intuitively natural and has a reasonable running time Reasons for using our approach versus previous methods are also sketched

1 Introduction

Many problems in data base management and information retrieval require the clustering of "objects" into classes such that objects within a class satisfy a certain property while objects across classes do not possess the property Some common examples are

(i) record clustering ([Jako, LiYa, OmSc]), in which records that are frequently jointly accessed should appear in the same page or block, while records in different blocks are rarely jointly accessed The objects in this problem are the records, the property satisfied by the objects within a class is "frequently jointly accessed"

(ii) file allocation in distributed data bases [Aper, Case, ChLi, DoFo, Eswa, FiHo, Ghos, Grbe, LoPo, MaRi, UrOI, YSLC] where files which are often jointly accessed should appear in the same sites

(iii) document clustering [Crof, Salt, Salo,Van, VanRi, Yu], in which "similar" documents should be placed within the same cluster

Previous approaches to the above problems have some or all of the following difficulties

(i) User query statistics have to be gathered, accumulated and classified

(ii) Significant inaccuracies are incurred during the process of estimating the costs of processing different queries Thus, the expected optimal solutions to a clustering problem could be very different from the actual optimal solutions

(iii) A long elapsed time is needed to determine the set of clusters

(iv) Ignoring or not making use of information provided by the users during the processing of users' queries

We propose an adaptive clustering approach that alleviates the above problems to a certain extent In the next section, we first give in more detail the difficulties of earlier approaches Then we outline our proposed solution The aim is to develop a number of tools available for adaptive information system design

2 Difficulties of earlier solutions

(i) User query statistics have to be accumulated and classified This may turn out to be an expensive process For example, in record partitioning, different queries may access different sets of records and there are potentially 2^n query types where n is the number of records Collecting, classifying and storing the statistical data associated with the different query types can be very expensive Common solutions to this problem have been (a) query statistics are assumed to be given, (b) there are a few dominating queries/transactions that other transactions can be neglected, (c) the queries are restricted to certain special types, for example, conjunctive or disjunctive queries

* This research is supported in part by N S F under grant no IST-8302237

Unfortunately, these solutions are not entirely satisfactory for the following reasons (a) Even if query statistics are known, a record clustering procedure that manipulates statistics involving potentially exponential number of query types can be expensive in time and space (b) As query languages become friendlier, there will be more casual users and the effects of queries submitted by these users can be so significant that they cannot be ignored (c) Restricting query types is not satisfactory to users

(ii) Estimating costs for processing queries usually incurs significant inaccuracies For example, in [HaCh, HaNi], the cost for processing a query is based on the assumptions that (a) there are equal numbers of records satisfying the conditions $A_i = c_i$ for different constants c_i, where A_i is the i^{th} attribute,(b) the distributions of values in different attributes are independent and (c) records are distributed randomly in blocks It has been pointed out in [Chri] that such uniform and independent assumptions are not realistic

(iii) A significant time is required to determine the clusters and therefore the supporting file structure is not as responsive to users' changing needs as desired

(iv) Most of the clustering algorithms cluster objects based on syntactic relationships between the objects only (For example, in document clustering, information about co-occurrence data is used only) Important information which can be extracted from interactions with users and can probably be used to yield better clusters is ignored

3 Our Approach

Our approach to cluster objects is illustrated by figure 1

An incoming query is processed by the information system After its processing, the data requirements of the query, including some intermediate results, are obtained (For example, in record clustering, the requirements consist of the set of records retrieved by the query and the number of blocks containing the retrieved records) This information is passed from the information system to our algorithm, which uses it to update a certain "abstract structure" Unlike a real structure which contains contents of records and their relationships, the abstract structure provides information about which objects should belong to which class(es) only The abstract structure is modified by our algorithm for each incoming query so that the effect of all users' queries, up to and including the present query is incorporated into the structure After many user queries have been processed, the abstract structure will be examined If it is sufficiently similar in the memberships of objects in classes to the present actual information structure, then there is no need to change the actual structure, otherwise a procedure to modify the actual structure from the abstract structure is invoked Thus, our method can be considered as a reorganization process However, the problems we study here are very different from those concerned with finding reorganization points [Bato, Shne, Tuel, YaDT]

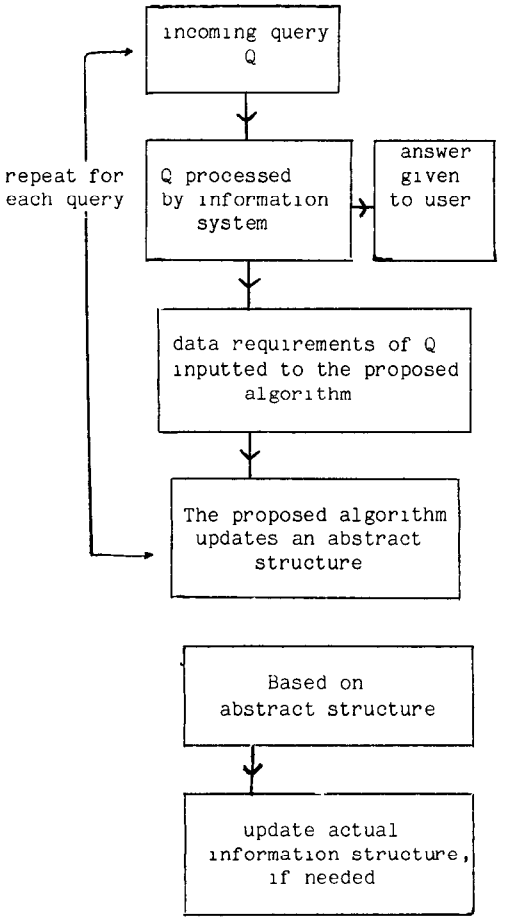


Figure 1 Proposed adaptive algorithm

From the above discussion, it is clear that there is no need to accumulate and classify query statistics The data requirements of each query is used by our algorithm to update the abstract structure After the update, the data requirements of the query are not stored, as their effect is incorporated into the abstract structure As data requirements are obtained by the information system during the processing of the query, estimation of the cost of processing the query is not needed Thus, inaccuracies due to such estimations by earlier methods are reduced or eliminated (It should be observed that it may not be feasible to store the data requirements for each query as the total amount of information to be stored for numerous queries will be extremely high)

The proposed algorithm performs a small amount of action for each query instead of performing a one-shot task involving a huge amount of work, after collecting and classifying statistics for many queries, as is usual for other algorithms As a result, the delay in determining a set of clusters for the given objects after processing the last query is very small for the proposed algorithm relative to other algorithms

In applications like document retrieval ([Salt, VanR]), each retrieved document in response to a query can be judged by the user to be relevant or irrelevant This information can be fed back to the system to allow the system to form better clusters

Thus, the four disadvantages of other approaches are alleviated by making use of the present proposed algorithm

In the next two sections, we give two adaptive clustering techniques to illustrate how the proposed algorithm can be used in practice

4 Adaptive clustering technique 1

This technique will be illustrated by our solution to the record clustering problem The problem is to assign records to blocks in such a way that the total number of blocks accessed, in response to a set of queries, is minimized Essentially the problem corresponds to assigning frequently jointly accessed records to the same blocks so that as few blocks as possible are

accessed per query More precisely, let $P(Q_1)$ be the probability that Q_1 is submitted and $Block(Q_1)$ be the number of blocks retrieved in order to answer Q_1 The cost to be minimized is

$$\sum_{Q_1 \in Q} P(Q_1) * Block(Q_1)$$

where Q is the set of all submitted user queries We assume that all records are of the same length and no record is duplicated This problem has been shown to be NP-hard [YSLT, OmSc]

Our initial solution to the problem is to propose an algorithm to a simplified record clustering problem Then, the algorithm will be modified to be applicable to the actual record clustering problem

The simplified record clustering problem is as follows

There are N records $\{R_1, R_2, \ldots, R_n\}$

Queries are issued to access some of these records The N records are to be partitioned into clusters such that frequently jointly accessed records appear in the same clusters while rarely jointly accessed records are in different clusters

A simple algorithm to solve the above problem is as follows

A position in the line $(-\infty,\infty)$ is assigned to each record As each query is processed by the data base management system, the positions of the records in the answer of the query are modified (The actual records in the data base are not moved) The cummulated effect of the modifications for many queries is that the positions of the records in the same cluster will be close together while the positions of records in different clusters will be far apart This allows the clusters to be easily identified and the records are then assigned to blocks There is no need to collect query statistics since the significance of query statistics is reflected by the relative positions of the records

Step(0) Initialization

The ith record R_i is initially assigned an arbitrary position X_i in a line for $1 \le i \le N$

Step(1) Moving Records Together

Let $\{Rm(i), 1 \le i \le k\}$ be the set of records in the answer of query Qm, $k \neq 1$, $k \neq N$ Let their positions be $\{Xm(i), 1 \le i \le k\}$ These k records are moved closer to their centroid by the amounts $DXm(i)$, $1 \le i \le k$, which will be determined later

The special cases for k=1 and k=N are handled as follows For a query accessing all records, any assignment of records to blocks will cause the same number of blocks to be retrieved Thus, there is no need to apply the above operation The same is true for a query accessing only one record

The centroid of the accessed records is

$$CX = \left[\sum_{i=1}^{k} Xm(i)\right] / k \qquad (4.1)$$

The distance moved by each accessed record is proportional to its distance from the centroid Specifically, the distance moved by the ith accessed record, $DXm(i)$, $1 \le i \le k$, is

$$DXm(i) = A * | Xm(i) - CX | \qquad (4.2)$$

where A is a coefficient governing the speed of movement, $0 < A < 1$

Suppose the accessed records move a constant distance instead of the distance as given in (4.2) Since the records are initially assigned arbitrary positions, it is possible that records belong to the same cluster are assigned extreme positions at the two ends of the line As a result, the convergence of the algorithm may be slow By (4.2), records that are far away are moved with larger distances than records that are close together, therefore compensating the random assignment of recores in

Step(0)

It is clear that if a set of records are often jointly accessed together, their corresponding points will converge However, the points corresponding to all the records may eventually bunch up together if Step(1) is repeated for many different queries To counter that, the following step is applied immediately after Step(1) is executed

Step(2) Pulling Records Away

Select k records randomly and move each of them apart from their centroid by an amount which is

A * (the distance of the record from the centroid of the k random records)

Steps(1) and (2) are executed for each query until a clear pattern of clusters can be observed (details are in [YSLS])

With the cost of main memory rapidly dropping, it is possible to store large amounts of information in main memory Thus, the positions of the records can be stored and updated in main memory

A sketch of analysis is as follows

Let c be the number of clusters, n_i be the number of records in the i^{th} cluster, $1 \le i \le c$ and X_{ij} be the position of the j^{th} record in the i^{th} cluster, $1 \le j \le n_i$ Let D_i be the centroid of the records in the i^{th} cluster The within distance W is defined to be

$$\sum_{i=1}^{c} W_i = \sum_{i=1}^{c} \sum_{j=1}^{n_i} (X_{ij} - D_i)^2$$

Clearly, $W \ge 0$ If W=0, then each record within the same cluster is located at the centroid of the cluster The between distance B is defined to be

$$\sum_{i=1}^{c} n_i (D - D_i)^2$$

where D is the centroid of all records If B is large, records of different clusters are far apart If W is small and B is large, records of the same cluster will be close together but far away from records of different clusters, making identification of the clusters an easy task Instead of using two different symbols, a sirgle measure W/B is used to indicate the ease of identifying the records in the clusters We seek a small value of the ratio

It is shown under certain reasonable assumptions in [YSLS] that if

$$A < [2b(W/B)]/[(d+b)W/B + d], \qquad (4\ 3)$$

then W/B is expected to decrease for each execution of a query, where b and d are certain positive constants The interpretation is as follows

For a given set of records at positions $\{X_{11}, X_{12}, \ldots\}$, (where X_{ij} denotes the position of jth record of the ith cluster) the values of W and B are defined For a chosen A, defined in Step (1) of the algorithm, if inequality (4 3) is satisfied, then W/B is expected to decrease for each execution of the query When W/B decreases, it can easily be verified that the R H S of (4 3) decreases If (4 3) is again satisfied, then W/B will continue to decrease This process continues until (4 3) is violated At that point,

$$W/B < Ad/[2b - A(d+b)] \qquad (4\ 4)$$

If the right hand side of (4 4) is small, then W/B will be small, allowing the identification of the clusters If the right hand side is not small, then W/B needs to be decreased This can be achieved by setting a smaller A than the one used earlier such that (4 4) is again satisfied The execution of more queries using the new value of A will allow W/B to decrease further Thus, the whole process consists of arbitrary choosing a value of $A < 1$, applying the algorithm for a set of queries until W/B does not decrease any further, reducing A and repeating these steps until W/B is sufficiently small to allow the records to be easily assigned to clusters It remains to identify the situations where A needs to be decreased (i e inequality (4 3) is violated) Since we do not know the actual value of W/B, the following heuristic is suggested

We compute the sum of the distances moved by the accessed records in Step(1) for a set of many queries, and the average is taken Let the average be denoted as TXi for time interval Ti (In our experiment to be described latter in this section, 400 queries are executed in each interval) Initially, when clusters are not formed, the records move large distances, since records belonging to the same clusters may be far apart Thus, initially Txi is large Gradually, when clusters begin to emerge, the records move smaller distances and Txi is smaller In other words, when cluster formation improves, Txi decreases On the other hand, if cluster formation deteriorates, we expect Txi to increase Thus, a criterion for adjusting the parameter A is as follows If $Tx(i-1) > Txi$, then clustering formation is improving, suggesting that the parameter A is appropriate There is no need to change A If the inequality is violated, then A is decreased, say set new A = 1/2*old A This process of adjusting A continues, until A becomes sufficiently small or the clusters are clearly formed (Details can be found in [YSLS])

We now suggest modifications to the proposed algorithm Step (2) of the algorithm is to ensure that all records are not bunched up together However, for each query, an additional k records need to be randomly selected In order to make the algorithm cheaper to implement, we simply

shift the accessed records away from the centroid of all records This variation is denoted as Version 2

Version 2 Shift Accessed Records Away

If the centroid of the accessed records is less than the controid of all records, then each of the accessed records is moved to the left by a distance equal to the average distance moved by the k accessed records Otherwise the direction of movement is to the right The intention of Version 2 is to shift the centroid of the accessed records away from the centroid of all the records such that the accessed records stay close together but far away from other records

Clearly, Steps (1) and (2) can be combined together into a single step, because the records to be modified are identical in both steps

The simplified record clustering problem does not take into consideration the cost, in terms of the number of blocks accessed, of processing the given set of users' queries Step (1) is now modified to incorporate the cost function The distance moved by the i^{th} accessed record, DXm(i), $1 \leq i \leq k$ is

$$DXm(i) = A * |Xm(i) - CX| * Benefit(Qm)/BENmax$$

where Benefit(Qm) is the benefit of placing the k records in the same cluster and be assigned to the minimum number of blocks over the current assignment of records to blocks, BENmax is the maximum benefit for any query and is used to ensure that Benefit (Qm)/BENmax ≤ 1

Experimental results comparing our proposed algorithm with the total sort method given in [Jako] are given as follows

Various problem sizes ranging from 100 records to 600 records are used to test the speed of convergence and the performance of the algorithm The number of queries is arbitrarily assumed to be the logarithm with base e of the number of records then multiplied by a certain value CQ The number of queries, the number of records and the CQ values used in the experiments are listed in the Table A

Query Set	CQ	Number of Records					
		100	200	300	400	500	600
A,B	6	27	31	34	35	37	38
C,D,E	6 5	34	34	37	39	40	42

Number of Queries

Table A Data Set

The qualification of each query in each query data is either in conjunctive (i e Ai=aij AND Ak=akl where the attributes are denoted A_i, A_k, etc, and the constants are aij, akl, etc) or disjunctive form (i e Ai=aij OR Ak=akl)

Since most user queries are likely to be simple, we arbitrarily set the number of single attribute queries to be about 1/2 of the total number of queries The number of two-attribute queries in conjunctive form is about 3/10 of the total number of queries and the remaining queries are two-attribute queries in disjunctive form

Each query set (sets (A) - (E)) for each of the six sets of records has a different level of the number of blocks accessed per query Let random performance be defined to be the average number of blocks required to retrieve the records for each query, assuming that records are randomly assigned to blocks For the five query sets, the random performances range from 40% of the total number of blocks for query set (A) to 15% for data set (E) In other words, each query in set A on the average retrieves more records than a query in the other sets

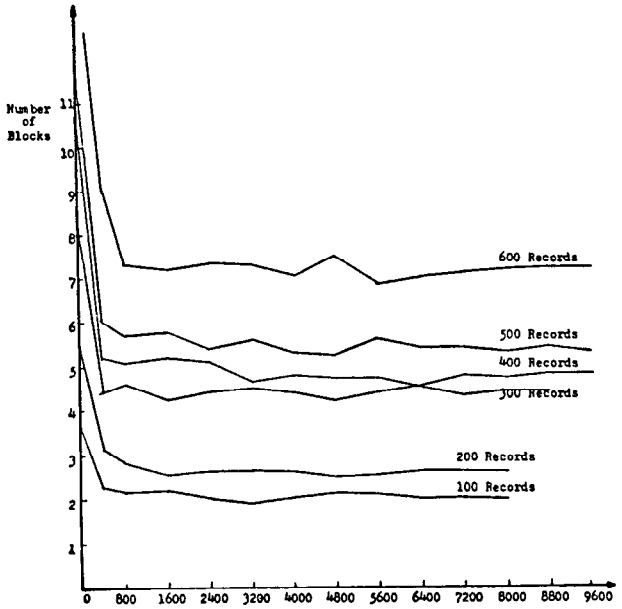
In order that the simulation is realistic, the access probabilities of the records form an approximate "80-20" [Knut] rule (i e approximately 80% of the accesses are directed toward 20% of the records) There are 4 attributes The block size for all sets of data is 10 records per block (As pointed out in [Jako], the results are not sensitive to block size That is why only one block size is used in the experiments) A typical result is given in Figure 2

Both versions of the algorithm converge extremely rapidly In fact, most of the gain in performance occurs over the processing of the first 400 queries From 400 queries to 3600 queries, further improvement in performance is obtained, though at a much slower rate From 3600 queries to the termination of the algorithm, there is very little change in performance

Let performance improvement percentage be computed as [performance of the total sort method-performance of version i] /performance of version i A summary of the results is as follows

1 Both versions of the proposed algorithm achieve better performance than the total sort method The improvement is particularly significant for Version 2

2 The average improvements of version 2 over the total sort method for the 5 query sets range from 34 56% to 72 61% When the number of records increases, or when the number of records accessed per query decreases, the improvement increases When the number of records is small, (in our experiments that is 100 records) Version 1 is better than Version 2 When the number of records is large, (in our experiments that is over 100 records) Version 2 achieves better performance than Version 1 Since Version 2 is more efficient in implementation and produces better results, we favor it



Number of Queries Processed Fig 2 Version 2, 30% Access of Total Number of Blocks (C) Query Set

5 Adaptive clustering technique 2

This technique will be illustrated by our solution to the problem of file allocations for distributed databases The problem is to assign files to sites such that the total communication cost for a given set of queries is minimized However, unlike earlier solutions [Case, FiHo, Ghos, etc], our approach takes into consideration the interaction between files during the processing of queries It has been pointed out in [YSLC] that such interactions between files are common and significant in current distributed query processing stratagies [SDD1, HeYa, YCTB, etc] (An illustration of the interaction between 2 files for processing a query is given in the Appendix) Apers [Aper] has considered such interactions Our approach differs from his in (a) Our method is the adaptive one-query-at-a-time approach with its associated advantages as outlined in Section 2 while his algorithm is a one-shot algorithm, (b) the number of variables in Apers' model may be too large to manage when files are replicated, (c) the problem formulations and the solutions are completely different

Let n be the number of files and m be the number of sites in a computer network Let y_{is} be a binary variable such that $y_{is} = 1$ represents the presence of file i at site s, 0 otherwise Thus, a file allocation is an assignment of variables

$$Y = \{y_{is} \mid 1 \le i \le n,\ 1 \le s \le m\} \rightarrow [0,1]$$

Let Q be a set of retrieval and update queries The file allocation problem is to find an assignment Lp such that the cost of processing the set of queries is minimized The cost is in terms of data communication cost between sites in order to answer retrieval queries or to synchronize copies of data due to concurrency control Clearly, the cost depends on the query processing algorithm and the concurrency control mechanism to be employed Given the query processing and the concurrency control algorithms, let the cost of processing a query q with a given file assignment be denoted by C(q|L) or C(q|L(Y)) It is clear that when a query q is processed, only certain variables in Y will affect the cost while changes in other variables in Y have no effect on the cost This subset of variables having direct effect on the cost of processing the query will be defined as q-associated variable set and denoted by Y_q

Whenever a query is processed, certain information will be computed immediately and accumulated with previous information in a structure, as described in the last two sections The structure used here is a n by m matrix A such that each entry A_{is} represents the desirability of assigning y_{is} to 1 (i e the allocation of file i to site s) Let the current file assignment be L_k When a query q is processed, the q-associated set, Y_q, is identified For each y_{is} in Y_q, we compute $B(q, y_{is}, L_k) = C(q \mid y_{is} = 0, L_k (Y_q - \{y_{is}\})) - C(q \mid y_{is} = 1, L_k (Y_q - \{y_{is}\}))$ and add it to A_{is} $B(q, y_{is}, L_k)$ represents the benefit of assigning file i to site s as compared to not assigning file i to site s for processing the query under the current assignment L_k After a set of queries is processed, A_{is} will be the aggregate benefit of assigning file i to site s for processing the set of queries

A sketch of the algorithm (with a given file assignment L_k) is as follows

Step 1
Initialization
$A_{is} = 0,\ 1 \le i \le n, 1 \le s \le m$

Step 2
For each processed query q,
for each y_{is} in Y_q

$A_{is} \leftarrow A_{is} + B(q, y_{is}, L_k)$

Step 3
After a set of queries Q is processed, determine a tentative file assignment L_k' by

$$L_k'(y_{is}) = \begin{cases} 1 & A_{is} > 0 \\ L_k(y_{is}) & A_{is} = 0 \\ 0 & A_{is} < 0 \end{cases}$$

Step 4

If $L_k' = L_k$, then {the same file assignment will be used and the algorithm terminates} else {let Y_d be the set of variables such that $L_k'(y) \neq L_k(y)$ Choose the y_{jt} in Y_d such that its corresponding $|A_{jt}|$ is maximum Construct L_{k+1}

$$L_{k+1}(y) = \begin{cases} L_k(y) & y \neq y_{jt} \\ L_k'(y) & y = y_{jt} \end{cases}$$

L_{k+1} is the current file assignment }

Steps (1-4) are then repeated

The following results can be proved [Chen]

(1) If the file allocation at the (k+1)th iteration is different from that of the k-th iteration, then the cost of the former allocation is strictly less than that of the latter allocation

(2) If there is an 1-neighbor of L_k (differs from L_k by the value of 1 variable only) whose cost is strictly less than that of L_k, then the algorithm will not stop at the (k+1)th iteration

(3) If an optimal solution is obtained at the k-th iteration, then the algorithm will terminate with the optimal solution at the (k+1)th iteration

The above algorithm assumes an initial allocation This allocation can be computed as follows

Step 1 Initialization

$A_{is} = 0$, $1 \le i \le n$, $1 \le s \le m$

Step 2 For each processed query q,

For each y_{is} in Y_q

A_{is} <--- A_{is} + 1/2 MinB(q,y_{is}) + 1/2 MaxB(q,y_{is}) where MinB(q,y_{is}) and MaxB(q,y_{is}) are respectively the minimum and the maximum benefits of assigning file i to site s among all possible assignments These quantities can easily be computed

Step 3 After a set of queries Q is processed, determine the initial file assignment L_o

$$L_o(y_{is}) = \begin{cases} 1 & \text{if } A_{is} > 0 \\ 0 & \text{if } A_{is} \le 0 \end{cases}$$

It can be shown that the initial file allocation satisfies the following desirable properties

A necessary condition and a sufficient condition for allocating a file to site s can be determined The conditions are generalizations of those given in [GrBe] incorporating the interactions between files (i e if interactions between files are neglected, the results are the same as those given in [GrBe]) The initial file assignment satisfies the following property

(1) If the sufficient condition for allocating file i at sites is satisfied, then the algorithm will allocate file i at site s without checking the truth or falsity of the sufficient condition

(2) If the necessary condition to optimally allocate file i at site s is violated then the algorithm will not allocate file i at site s without checking the necessary condition

Thus, the initial file allocation is reasonable

Experiments are performed on two versions of the problem In the first version, a file can be assigned to any site with the constraint that it must be assigned to at least one site In the second version, each file is assigned to a pre-selected site, due to ownership or administration Additional copies of the file may be assigned to other sites The first version is called FAP without ownership and the second version is called FAP with ownership

Retrieval queries, each accessing data from either one file or two files are randomly generated such that the access probabilities of files form an approximate Zipfian distribution with a random parameter O_1 Similarly, update queries are generated such that the update probabilities of files form the Zipfian distribution with another random parameter O_2 Each query type can be submitted from every site and the distribution of the probabilities of each query type among the sites form another Zipfian distribution with parameter O_3

Two sets of experiments are run The first set of experiments is designed to test the accuracy of the algorithm Small problem sizes ranging from (3 sites, 6 files) to (4 sites, 10 files) are used so that a branch and bound algorithm obtaining optimal solutions will terminate In each problem size, 180 cases are run and the average is taken The error rate for version 1 of the problem (FAP without ownership) is given in Table 1 In each problem size, the average relative error is less than 50% (This seems to compare favorably with the result in [Aper1] where an average error for 15 cases is reported to be 3% However, the problem and the test cases are not identical) Table2 shows the error rate for the FAP with ownership The error is much smaller, with no entry larger than 04% There is some increase in error rate when the number of sites is increased from 3 to 4 However, when the number of sites is increased to 5 and the number of files is 6, there is a drop in error rate (The figure is not given here The branch-and-bound optimal algorithm does not terminate for larger problem sizes)

The second set of experiments is designed to test the speed of convergence First, the number of sites is fixed at 10 and the number of files is increased from 12 to 40 in intervals of 4 Then the number of files is set equal to 40 and the number of sites is increased from 3 to 24 in intervals of 3 20 cases for each problem size are run and the average is shown in Table 3 The result indicates that the average run time is

proportional to the number of files and the square of the number of sites (The average computing time, if projected to 58 sites and 1 file is less than 30 seconds This is likely to be better than the time of about 3 minutes reported in [FiHo] However, a direct comparison is not possible because different computers are used More importantly, the problems are different Their algorithms assign files optimally but without considering the interactions between files, while our algorithm takes such interactions into consideration

		Number of Files				
		6	7	8	9	10
Number of Sites	3	26	37	24	34	22
	4	38	29	33	42	33

Table 1 FAP without ownership error rate in %

	6	7	8	9	10
3	00	02	00	00	00
4	04	01	03	03	01

Table 2 FAP with ownership error rate in %

10 Sites

Number of Files

12	16	20	24
11 89	14 69	19 32	21 52

28	32	36	40
23 28	27 48	33 68	36 84

40 Files

Number of Sites

3	6	9	12
5 67	13 89	28 67	48 29

15	18	21	24
83 31	108 88	157 38	180 59

Table 3 Convergence rate (time in seconds) A VAX/750 is used

6 Conclusion

We believe the techniques presented here are applicable to many other problems, such as index selection [Chan,HaCh], IMS segment clustering [Schk] and document clustering [Salt,VanR,Yu] Essentially, these techniques aim at obtaining good but not necessarily optimal solutions With records or files accessed with highly uneven probabilities (e g "80-20" rule or the Zipf's distribution), the experimental results given in the previous sections seem to confirm the good performance of the heuristics It is hoped that the ideas presented here will be useful in constructing adaptive information systems

7 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, Feb 1981, pp 245-258

[Aper1] Apers, P M G , "Data allocation in distributed database systems", ACM TODS, (to appear)

[Bato] Batory, D S "Optimal File Designs and Reorganization Points", ACM TODS, 1982, pp 60-81

[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] Chan A Y "Index selection in a Self-Adaptive Relatimal Data Base Management System", M Sc thesis, Dept of EECS, M I T 1976

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

[Chen] Chen, C H , "An adaptive technique and its application to database design", Ph D dissertation, Dept of EECS, University of Illinois at Chicago, 1984

[Chri] Christodoulakis S "Estimating block transfers and join sizes", ACM SIGMOD, 1983, pp 40-45

[Cook] Cook, S A , "The complexity of theorem proving procedures", ACM SIGACT, 1970, pp 151-158

[Crof] Croft, W B , "Clustering large files of documents using the single link method", JASIS, 1977, pp 341-344

[DoFo] Dowdy, L W and Foster, D V , "Comparative Models of the file Assignment problem", 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, Aug 1974, pp 304-307

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

[FUNG] Fung, K T "A reoganization model based on the database entropy concept," The Computer Journal, 1984, pp 67-71

[Ghos] Ghosh, S P , "Distributing a data base with logical assocations 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, Nov 1977, pp 878-882

[HaNi] Hammer, M and Niamir, B , "A heuristic approach to attribute partitioning", ACM SIGMOD, 1979, pp 93-101

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

[HeYa] Hevner A and Yao B "Query Processing in distributed database systems", IEEE Transactions on Software Engineering, 1979, pp 177-187

[HoSa] Horowitz, E and Sahni, S , "Fundamental of Computer algorithms", Computer Science Press Inc , 1978

[Jako] Jakobsson, M , "Reducing block accesses in inverted files for partial clustering", Information Systems, 1980, pp. 1-5

[Karp] Karp, R "Reducibility among combinational problems", In Complexity of Computer Computations, R Miller and J Thather Ed , Plenum Press, N Y 1972, pp 85-104

[Knut] Knuth, D., "Sorting and Searching", Vol 3 , Addison Wesley, 1973

[LaYu] Lam, K and Yu, C T , "A clustered search algorithm incorporating arbitrary term dependencies", ACM TODS, Sept 1982, pp 500-508

[LiYa] J K Liou and S B Yao "Multi-Dimensional Clustering for Data Base Organization" Information Systems, Vol 2 pp 187-198 (1977)

[LoPo] Loomis, M S and Popek, G J , "A model for database distribution", Trends and Applications, Computer Methods, ~ IFEE Computer Society, 1976

[MaRi] Mahmoud, S and Riordon, J S , "Optimal allocation of resourses in distributed information networks", ACM TODS, Vol 1, No 1, Mar 1976, pp 66-78

[MaSe] March, S T and Severance, D G , "The determination of efficient record segmentations and blocking factors for large shared databases" ACM TODS, 1977, pp 279-296

[OmSc] E Omiecinski and P Scheuermann "A Global Approach to Record Clustering and File Reorganization", Technical Report, Dept of EECS, Northwestern Univ , (Dec 1983)

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

[Schk] Schkolnich M "A Clustering Algorithm for Hierarchical Structures", ACM TODS, 1977, pp 27-44

[Shne] Shneiderman B , "Optimal database reorganization points", CACM,1973, pp 362-365

[SDD1] Bernstein, P Goodman, N Wong E Reeve, C Rothnie J "Query Processing in a system for distributed databases", ACM TODS, 1981, pp 602-695

[Tuel] Tuel, W G , "Optimum reorganization points for linearly growing files", ACM TODS, 1978, pp 32-40

[Van] Van Rijsbergen, C J , "Information Retrieval", 2nd edition, Butterworths, London, 1979

[VanR] Van Rijsbergen, C J , "Further experiments with hierarchical clustering in document retrieval", Information Storage and Retrieval, 1974, pp 1-14

[Uro1] Urano, Y , Ono, K and Inone, S , "Optimal design of distributed networks", Proc of International Conference on Computer Communication, Aug 1974, pp 413-420

[Wah] Wah, B W , "An efficient heuristic for file placement on distributed database", IEEE COMPSAC 1980, pp 462-468

[Yao] Yao, S B , Das, K S , and Teorey, T J , "A dynamic database reorganization algorithm", ACM TODS, 1976, pp 159-174

[YCTB] Yu, C T Chang C Templeton, M and Brill D "On the design of processing a distributed query processing algorithm", ACM SIGMOD, 1983, pp 30-39

[YGCC] Yu, C T Guh, K Chang C C , Chen C H , Templeton, M and Brill D "Placement dependency and aggregate processing in a fragmented distributed database environment" IEEE COMPSAC, 1984 (to appear)

[YGCT] Yu, C T Guh, K , Chen, C , Chang C , Templeton M and Brill D "An algorithm to process queries in a fast distributed network", IEEE Real time systems symposium, 1984 (to appear)

[YSLC] Yu, C T , Siu, M K , Lam, K and Chen, C H , "File allocation in distributed databases with interaction between files",VLDB 1983, pp 248-259

[YSLS] Yu, C T Suen, C M , Lam K and Siu M K "Adaptive record clustering", Technical report, Dept of EECS, 1984

[YSLT] Yu, C T , Siu, M K , Lam, K and Tai, F , "Adaptive clustering Schemes General Framework", IEEE COMPSAC, Nov 1981, pp 81-89

[Yu] Yu, C T "A clustering technique based on user queries", JASIS, 1974, pp 218-226

Appendix

Let $_{ij}R_s$ be a query accessing files i and j (representing relations i and j respectively) We now describe how this query can be processed and define its cost in terms of different file allocations, under the assumption that the communication cost between two sites does not depend on the sending and the receiving sites This assumption is commonly used in distributed query processing algorithms [HeYa, SDD1, YCTB, etc]

(Case 1) There is no site containing both files i and j

(1 a) Neither file i nor file j is at site s

Semi-joins will be performed before file i and file j are sent to site s The semi-join sequence can be either that a semi-join from relation i to relation j across the sites is executed to reduce relation j followed by a semi-join from the reduced relation j to relation i or the above semi-join sequence in reverse order, depending on which sequence has less cost Let F_k (R) denote the size of the reduced file k and SF_k (R) be the size of file k projected on the join attribute and RSF_k (R) be the size of the reduced file k projected on the join attribute Note that $SF_k \geq RSF_k$ (R) The cost of processing the query $_{ij}R_s$ is (assuming unit data communication cost)

$$RC(_{ij}R_s \mid y_{is} = 0, y_{js} = 0, z_{ijs} = 0)$$

$$= \text{Min } \{SF_i (R) + RSF_j (R), SF_j (R) + RSF_i (R)\} + [F_i (R) + F_j (R)]$$

where the first two expressions denote the costs of the semi-joins sequences and the last expression denotes the cost to send the reduced relations to an assembly site to take the join, y_{ks} is a binary variable such that it is 1 if file k is situated at site s and 0 otherwise, z_{ijs} is a binary variable such that it is 1 if there is a site other than s containing both files i and j and 0 otherwise

It is possible that one or both of the semi-joins may not be executed, if they do not cause sufficient reduction of the sizes of the relations It is also possible that one of the reduced files is sent to the site containing the other file, the answer is constructed from these files and then it is sent to site s These situations can also be handled by our approach But for ease of presentation, we assume the above cost equation

(1 b) One of the two files, say file i, is at site s

The join attribute values of file i can be sent to a site containing file j to reduce the file, then the reduced file j is sent back to site s to join with file i to construct the result The other way is to send the join attributes values of file j to site s to reduce file i, then the join attributes values of the reduced file i are sent back to reduce file j and finally the reduced file j is sent to site s to join with the reduced file i So, the cost is

$$RC (_{ij}R_s \mid y_{is} = 1, y_{js} = 0, z_{ijs} = 0)$$

$$= \text{Min } \{SF_i (R), SF_j (R) + RSF_i (R)\} + F_j (R)$$

This means that file j is reduced by one or two semi-joins before it is sent to site s

Similarly, if only file j is at site s, the cost is

$$RC (_{ij}R_s \mid y_{is} = 0, y_{js} = 1, z_{ijs} = 0)$$

$$= \text{Min } \{SF_i(R) + RSF_j(R), SF_j(R)\} + F_i(R)$$

(Case 2) There is a site containing both files i and j

(2 a) There is a site other than site s, say site t, containing both files i and j In this situation, either the answer or the

reduced files which are absent from site s is sent to site s The cost of processing the query ij^Rs is

$$RC(ij^Rs \mid y_{is}, y_{js}, z_{ijs} = 1)$$

$$= \text{Min } \{F_{ij}(R), (1-y_{is}) \times F_i(R) + (1-y_{js}) \times F_j(R)\}$$

where $F_{ij}(R)$ is the size of the result of the query ij^Rs

(2 b) Both files i and j are at site s The cost of processing the query ij^Rs is zero no matter what the value of z_{ijs} is and it is represented as follows

$$RC(ij^Rs \mid y_{is} = 1, y_{js} = 1, z_{ijs}) = 0$$