INDEX SELECTION IN A SELF-ADAPTIVE DATA BASE MANAGEMENT SYSTEM

Michael Hammer
Arvola Chan

Laboratory for Computer Science, MIT,
Cambridge, Massachusetts, 02139.

We address the problem of automatically adjusting the physical organization of a data base to optimize its performance as its access requirements change. We describe the principles of the automatic index selection facility of a prototype self-adaptive data base management system that is currently under development. The importance of accurate usage model acquisition and data characteristics estimation is stressed. The statistics gathering mechanisms that are being incorporated into our prototype system are discussed. Exponential smoothing techniques are used for averaging statistics observed over different periods of time in order to predict future characteristics. An heuristic algorithm for selecting indices to match projected access requirements is presented. The cost model on which the decision procedure is based is flexible enough to incorporate the overhead costs of index creation, index storage and application program recompilation.

INTRODUCTION

The efficient utilization of a data base is highly dependent on the optimal matching of its physical organization to its access requirements and other characteristics (such as the distribution of values in it). We consider here the problem of automatically tuning the physical organization of an integrated data base. By an integrated data base, we mean one that supports a diversity of applications in an enterprise; the development of such data bases is expected to be one of the most important data processing activities for the rest of the 70's [1]. There are many reasons for the incorporation of heretofore separate but related data bases with a high degree of duplication into a single integrated one. The reduction of storage and updating costs, and the elimination of inconsistencies that may be caused by different copies of the data in different stages of updating, are among the more important ones. Viewing an integrated data base as the repository of information for running an enterprise, it can no longer be considered as a static entity. Instead, it must be looked upon as continually changing in size, with access requirements gradually altering as applications evolve, and as users develop familiarity with the system. Consequently, the tuning of a data base's physical organization must also be a continual process. In current data base management systems, the responsibility of making reorganization decisions falls on the data base administrator (DBA), whose judgements are based on intuition and on a limited amount of communication with some individual data base users. For large integrated data bases, a more systematic means for acquiring information about data base usage, and a more algorithmic way of evaluating the costs of alternative configurations, will be essential. A minimal capability of a data base management system should be the incorporation of monitoring mechanisms that collect usage statistics while performing query processing. A more sophisticated system would sense the change in access requirements, evaluate the cost/benefits of various reorganization strategies, and recommend action to the DBA; eventually, such a system might itself perform the necessary tuning.

INDEX SELECTION IN AN ADAPTIVE DATA BASE SYSTEM

We are currently developing a self-adaptive data base management system which monitors the access patterns and the data characteristics of a data base, and uses this information to tune its physical organization. We operate in the environment of a relational data base system, which provides a level of physical data independence that facilitates physical reorganization. Continuous monitoring of the usage of a relational data base opens up many possibilities for its reorganization, and we expect to experiment with a variety of alternatives and study their costs and tradeoffs. As a first cut at the problem, we have concentrated on the problem of index selection. A secondary index (sometimes referred to as an inversion) is a well-known software structure which can improve the performance of accesses to a relation (file) [1]. For each domain (field) of the relation that is inverted, a table is maintained, which for each value of the domain in question contains pointers to all those tuples (records) whose contents in the designated domain is the specified value. Clearly, the presence of a secondary index for a particular domain can improve the execution of many queries that reference that domain; on the other hand, maintenance of such an index has costs that slow down the performance of data base updates, insertions, and deletions. Roughly speaking, a domain that is referenced frequently relative to its modification is a good

candidate for index maintenance. The choice of which (if any) domains to invert must be done with care; a good choice can significantly improve the performance of the system, while a bad selection can seriously degrade it. The goal of our system is to make a good choice of those domains for which to maintain secondary indices, based on how the data base is actually used.

The operation of the initial version of our prototype system can be described as follows. The specifications of data base interactions, by both interactive users and application programs, are expressed in a non-procedural language; these are first translated into an internal representation made up of calls to system level modules. The language processor has available to it a model of the current state of the data base, which contains, among other things, a list of the currently maintained set of secondary indices, plus various information about these indices. Using this information, the language processor can choose the best strategy for processing each data base operation in the current environment. Statistics gathering mechanisms are embedded within the system modules that interpret the object code of the language processor; and record data concerning the execution of every data base transaction. The statistical information gathered for a run is deposited in a collection area and summarized from time to time. When the reorganization component of the system is invoked (which will occur at fixed intervals of time), the statistical information collected over the preceding interval is combined with statistics from previous intervals and used to obtain a forecast of the access requirements of the upcoming interval; in addition, a projected assessment of various characteristics of the data in the data base is made. A near-optimal set of domains for which indices should be maintained is then determined heuristically; optimality means with respect to total cost, taking into account index storage and maintenance. This cost is compared with the projected cost for the existing set of indices. Reorganization is performed only if its payoff is great enough to cover its cost as well as that of application program retranslation.

In this paper, we stress our approach to the problem of acquiring an accurate usage model and estimating data characteristics, by means of continuous monitoring and the application of forecasting techniques. We have considered the reduced problem of choosing indices for a single relation, but we expect our approach to be extendible to more complex situations once we have developed a model for the cost of processing multi-relation queries using different strategies. We believe that an accurate interaction and cost model is essential for a practical environment, and that the use of problem-oriented heuristics to cut down the index search space will be fruitful in achieving efficient and near-optimum solution procedures.

ORGANIZATION OF THE PAPER

The remainder of this paper is organized as follows. We begin with a summary of our view of the data base organization: the file model, the storage and index organizations, and the data base operations. Next, a procedure for determining the strategy (whether or not to use existing indices) for processing the qualification part of a query is presented. We then describe the statistics that are to be gathered during query processing, and explain the use of exponential smoothing techniques in the derivation of parameters for the cost model. This is followed by a discussion of the need for heuristics in solving the index selection problem, and then the heuristics that we have devised are presented. Finally, we include a brief comparison with previous related work.

FILE MODEL

We have chosen the relational model [2] of data as the logical interface between application programs or interactive users and the data base system, since it provides the level of physical data independence needed to facilitate physical reorganization. The totality of formatted data in the data base consists therefore of one or more relations. However, we address here the reduced problem of selecting indices for a data base made up of a single relation. Herein, the data base consists of a single relation (file) R with n tuples (records) $T = (t_1, t_2, \cdots, t_m)$ where each $t_i \in D_i$, the ith domain, i.e. R is a subset of the cartesian product of the domains $D_1, D_2, \cdots, D_m$. Even though insertion and deletion of tuples are permitted, we will assume that the cardinality (number of tuples) of the relation remains relatively unchanged between two consecutive points at which index selection is considered.

Following Rothnie [3], we assume a paged memory environment for tuple storage. More specificly, the n tuples in the relation R are stored in p pages of t tuples each, where t is constant and n and p approximately so. The accessing cost of a page is assumed to be independent of the sequence of page accesses, and dominates all other internal processing costs. Hence, the processing cost for a query is measured solely in terms of the number of pages that have to be accessed to resolve the query. If a query is resolved by sequentially scanning all tuples in the relation, then the total number of pages that are accessed is just p. If an index is available for a domain referenced in a query, it may significantly reduce the number of tuples that need to be examined to resolve the query. These tuples may correspondingly reside on fewer than p pages. We shall assume that such a restricted set of tuples will be randomly distributed over the memory space, but that it is possible to access them such that each page is touched at most once. Yue and Wong [4] have derived an exact formula for the expected number of pages that have to be touched in order to access r tuples which are randomly distributed over the file. Let b(r;p,t) be the expected number of page accesses for referencing r randomly distributed tuples in a file of p pages each containing t tuples, and let

$$f(0) = 0$$
$$f(i+1) = ((t*(p-1)-i)/(p*t-i))*f(i) + (p*t)/(p*t-i)$$

then it has been shown in [4] that $b(r;p,t) = f(r)$. We will make use of the above formula to determine if it is profitable to use index(es) to resolve a query.

INDEX MODEL

An index on a domain of a relation is a mapping from values of the domain to tuples in the relation with those values. We assume that indices, just like tuples in the relation, are stored in a paged memory, with the usual two-level hierarchical organization, i.e. the key values and their associated tuple identifier (TID) lists are assumed to be on separate levels [5]. Each value is stored together with a pointer to the head of its associated TID list, and the length of that list. To further simplify our discussion here, we will assume that the keys and the associated list pointers and list lengths are organized as a B-tree [6], each node of which is stored on a page. One or more TID lists may be stored on the same page. The cost of obtaining the TID list associated with a domain value in the index is thus made up of the cost of locating the head of the list (equal to the height of the B-tree, which is dependent on the number of values in the domain), plus the cost of reading the page(s) on which the TID list is stored. Hence, the average cost for using an index can be readily estimated, given the number of values in the domain and the lengths of the associated TID lists.

The cost of modifying an index as the result of a tuple insert/delete/update is, however, more difficult to determine. We assume that only those domain values that are associated with existing tuples in the relation are stored in the index. In most cases, an insert/delete/update will cause a single node in the B-tree to be updated. In rarer cases, a key may have to be inserted into/deleted from the B-tree, resulting possibly in a node overflow/underflow, which may in turn propagate through the tree. As for the TID list involved, it is possible in many cases to write back a lengthened version of it onto the same page on which it was stored before the insertion, by utilizing empty spaces in the original page. Occasionally, though, the empty space may run out, and it becomes necessary to allocate a new page. Similarly, occasional garbage collection may be necessary to recompact the TID lists, and recover wasted space caused by deletions. Therefore, for a given domain, the cost of key insertion, key deletion and garbage collection depends on the sequence in which "updating" operations involving this domain are performed; therefore it is very difficult to incorporate this cost into any overall parametric index maintenance cost model. We will therefore consider the index maintenance cost for each domain as made up of two components. The first depends only on the frequencies of inserts/deletes/updates for this domain and consists of the cost of updating a node in the B-tree plus the cost of reading and writing an average TID list (an update is essentially equivalent to an delete and an insert); this component can be readily estimated. The second, made up of the unquantifiable overheads of node splitting/merging in the B-tree and garbage collection in the TID space, will be recorded by the statistics gathering mechanism if the domain is indexed. Otherwise, it can very roughly be estimated by using the normalized average overhead of those indices that are maintained.

TRANSACTION MODEL AND PROCESSING

Our transaction model allows for the retrieval, insertion, updating, and deletion of tuples in a relation. Data selection (the specification of some subset of a relation by means of its properties) is the fundamental component of all these operations. In choosing data selection operators to be included in our transaction, we have limited ourselves to those for which the utility of using indices can readily be estimated. These include conjunctions of equality conditions and disjunction of equality conditions. (By an equality condition, we mean a predicate of the form A = k, where A is some domain name and k is a constant or a program variable). The use of indices and list processing techniques for resolving these kinds of queries is well known [7], and will not be reiterated here. However, our tuple access cost model implies that it may not always be desirable to resolve a conjunction or a disjunction of equality conditions using indices. (This will occur when the set of qualified tuples is expected to reside on close to p pages; we say "close to", because utilizing the index(es) also entails some page accesses.) Therefore, we need a means whereby the number of tuples that can be expected to qualify for such a query can be estimated. We can define a selectivity measure for a domain index as the average fraction of the set of tuples under consideration that have historically satisfied an equality condition involving that domain. We will, furthermore, assume that the joint selectivity of a number of domains specified together is equal to the product of their individual selectivities. Hence, given the selectivities of domains in each condition, the number of tuples that can be expected to satisfy a conjunction of these conditions can be determined; in addition, the number of tuples for a disjunction can be estimated using the inclusion-exclusion principle [8]. Thus, given a query and an existing set of indices, the query processor will use the available indices only if the expected sum of the cost of accessing the indices and the resolved list of tuples is less than the cost of doing a sequential search of the entire tuple space.

STATISTICS GATHERING

An important component of our adaptive system is the monitoring mechanisms that collect usage statistics as data base operations are performed. The statistics gathered over a time period are summarized and then "averaged" with summaries from previous periods. The statistics to be gathered for the purpose of index selection fall into three general classes.

(1) update related statistics - This has several components. First of all, there are the total numbers of tuples that are deleted from (inserted into) the relation in the current time period. (We assume that each insertion or deletion of a tuple will require some maintenance for each active index.) In addition, for each domain of the relation, we maintain the number of updates made to that domain in the tuples. (This involves some maintenance of the index for that domain.) Finally, we record the actual difficult-to-parameterize costs of the maintenance of each index (node splitting and merging, garbage collection, etc.), measured in terms of the actual number of page accesses expended for such overheads.

(2) domain selectivity statistics - For each domain, we maintain its average selectivity over all uses of the domain in equality conditions in the current interval. This is accomplished by recording the number of times the domain occurs in equality conditions and the selectivity of the domain in each of these predicates. The selectivity of the use of a domain in an equality condition is measured as follows:

(a) If an index for the domain is used to resolve the particular equality condition, then the precise selectivity of the domain for this query can be calculated as the fraction of tuples in the relation with the domain value in question.

(b) Suppose the equality condition appears in a conjunction of conditions that is resolved by sequential scanning. (This scan may be of a reduced set of tuples obtained via an index.) Let the total number of tuples scanned be N_0. Let the conjunction be of the form $C_1 \wedge C_2 \wedge \cdots \wedge C_n$ where each of the C_i is an equality condition involving domain D_i. Let $N_1, N_2, \cdots, N_n$ be the number of tuples that satisfy C_1, $C_1 \wedge C_2$, $\cdots$, $C_1 \wedge C_2 \wedge \cdots \wedge C_n$ respectively. (Note that these values are readily available). The selectivity of domain D_i for this query is then approximated as N_i/N_{i-1}.

(c) Suppose the equality condition appears in a disjunction of conditions that is resolved by sequential scanning. Let the total number of tuples scanned be N_0. Let the disjunction be of the form $C_1 \vee C_2 \vee \cdots \vee C_n$ where each of the C_i is an equality condition involving domain D_i. Let $N_1, N_2, \cdots, N_n$ be the number of tuples that satisfy C_1, $\sim C_1 \wedge C_2$, $\cdots$, $\sim C_1 \wedge \sim C_2 \wedge \cdots \wedge C_n$ respectively. (Note that these values are readily available). The selectivity of domain D_i for this query is then approximated as $N_i/(N_0 - \text{Sum}(N_j) \text{ for } j < i)$.

Note that this measurement of selectivity accounts for non-uniform distribution of domain values over the tuples, as well as the non-uniform use of domain values in the equality predicates.

(3) query type statistics - Every time a query is processed, its associated qualifying expression is recorded in an access history file using a canonical representation (such as a bit coding scheme for each domain that appears in the qualification expression and a bit indicating whether it is a conjunction or a disjunction). Note that by recording the actual queries that occur, we avoid the strong (and often inaccurate) assumption that the simultaneous occurrence of domains in a query are mutually independent events.

ACCESS PATTERN FORECASTING AND PARAMETER ESTIMATION

We assume that the index selection problem is to be reconsidered at fixed intervals. At each reorganizational point, we forecast a number of characteristics for the period up to the next reorganizational point. Specifically, we predict the following:

(1) the number of occurrences of each query type (the type of a query is determined by whether its qualification component is a conjunction or a disjunction, and by the domains used in the constituent equality conditions);

(2) the selectivity of each domain of the relation;

(3) the number of distinct values in each domain (if a domain is not indexed in the current interval, then the current value for this number is approximated as the reciprocal of the observed selectivity);

(4) the number of tuples (hence the number of pages) in the entire relation;

(5) the expected cost (in terms of page accesses) of maintaining an index for each domain and the expected cost of each use of such an index.

We could do these projections solely on the basis of statistics collected during the previous time period, or we could combine together the statistics collected over all previous periods. However, neither alone would be

satisfactory for the purpose of a stable and yet responsive adaptive system. In the former case, the system would be overly vulnerable to chance fluctuations in access requirements and data characteristics, whereas in the latter case, it would be too insensitive to real changes. Intuitively, a weighted moving average of observations over previous periods should be more satisfactory. We utilize the technique of exponential smoothing [9,10] for our forecasting and estimation procedures because of its simplicity of computation, its flexibility, its minimal storage requirement, and its ability to be generalized to account for trends and cycles. The basic formulation of exponential smoothing in making a forecast of a discrete time series is as follows:

new forecast = α * (actual observation from last period) + (1-α) * (previous forecast)

where α is called a smoothing constant and takes on values between 0 and 1. In essence, this is a weighted average of all previous observations with the weight decreasing geometrically over successively earlier observations. The rate of response to recent changes can be adjusted simply by changing the smoothing constant: the larger the smoothing constant, the more sensitive is the forecast to recent changes and chance fluctuations. (The value of α can be selected by the DBA or can be adaptively chosen by the system itself to minimize the difference between observed and predicted data.) The compactness of the scheme lies in the fact that only two parameters need to be maintained for each time series, the current observation and the previous estimate. The above scheme is used for the estimation of domain selectivity.

Since the file size can generally be expected to be growing and the level of activity in a data base system can generally be expected to be upward moving as users become familiar with the capabilities of the system, we will use a modified version of exponential smoothing, which takes trends into account, in order to forecast query frequencies, index maintenance cost, and file size. Its formulation is as follows [9]:

new average = α * (current observation) + (1-α) * (old average)

current trend = new average - old average

new trend = α * (current trend) + (1-α) * (old trend)

new forecast = new average + ((1-α)/α) * (new trend)

In this form, it is necessary to store only the previously calculated values for the (new) average and for the (new) trend, and the calculation is still simple.

INDEX SELECTION AT A REORGANIZATION POINT

As we have said, at each reorganizational point, a number of forecasts are made. A projection is made of the selectivity of each domain in the relation, together with the expected costs of accessing and maintaining an index on that domain during the next interval. In addition, the expected size, and hence the storage requirement, of an index for each domain is estimated. Then the cost of creation for each index that is not currently maintained is approximated. For any proposed set of indices, we can thus project a total cost for the next time period. (This cost includes retrieval processing; index creation, maintenance and storage; and application program recompilation.) Therefore, we should be able to choose that set of domains whose indexing in the next interval will minimize this total cost.

A straightforward approach to the index selection problem would be to evaluate the total cost for each possible choice of domains to be indexed, and then select that set of domains which gives the smallest cost. With m domains in the relation, there are 2^m possible choices of index sets. For small m, this enumerative approach is probably the best strategy, as the optimal combination of domains to be indexed is guaranteed to be found. But for moderate m, the cost of repeated application of the cost evaluation procedure becomes very expensive, and for large m, it is prohibitive. It is not uncommon to find single-relation data bases with tens of domains. Therefore, it is appropriate to look for ways whereby the search space of potential index sets can be systematically reduced. There are two possible approaches to this. One is to look for properties in the cost function that will allow it to be minimized without exhaustive enumeration, such as through a depth-first search. This is the approach that Schkolnick [11] has taken. However, even with a simplified model for cost evaluation, his upper bound of the order of $2^{m^{0.5}\log m}$ sets to be tested is not enough of a reduction to enable the inexpensive selection of the optimal index set for a relation with a moderate number of domains. If a more accurate and detailed cost model is to be used, then the incentive for reducing the search space is even stronger. Yet in this case, the hope of finding an algorithmic way of exploring a reduced search space of practical size that will yield the optimal solution is even dimmer. Therefore, it is appropriate to draw on the experience of artificial intelligence researchers working in areas where formal mathematical structures are computationally impractical [12,13], and use heuristic methods that significantly prune down the search space and that work towards obtaining a near-optimal solution.

INDEX SELECTION HEURISTICS

In our index selection procedure, we make use of five problem-oriented heuristics.

(1) A near optimum choice of domains to be indexed can be made incrementally using a depth-first search. This heuristic permits analysis of the problem as a stepwise minimization, each time adding to the index set that domain which will bring the best improvement to the cost function. There have been two previous suggestions regarding the incremental selection of domains to be indexed. Farley and Schuster [14] suggest that the incremental selection process can be terminated once no single domain in the non-indexed set can be chosen that will yield incremental cost/benefits. This is insufficient for our choice of query and tuple access models. There are two reasons why it may be necessary to consider the incremental savings brought by considering two or more indices together. First, it may happen that for a query involving a conjunction of conditions, the selectivity of any one domain may not be sufficient to restrict the number of tuples to be accessed to be less than the total number of pages in the relation, whereas the joint selectivity of two or more domains might. Secondly, a disjunction of conditions can be resolved via indices only if all of the domains involved in the disjunction are indexed. An alternative strategy has been suggested by Held [15], who, at any stage of the incremental index selection procedure, considers the incremental savings of each of the possible subset of domains in the candidate set with less than or equal to some fixed number of domains in it. This, of course, may be very inefficient. We have taken an intermediate approach. We consider the adjoining of multiple domains to the index set only if no single domain that will yield positive incremental savings can be found.

(2) Some domains can be eliminated from the initial candidate set by virtue of their low occurrence frequencies in queries. This effectively reduces m, the initial number of domains in the candidate set. From the forecasted frequency of each retrieval specification, we can find the projected number of occurrences of each domain. Using the selectivity estimate of the domain, we can find a gross upper bound on the number of page accesses that an index on the domain can save in the processing of the forecasted set of queries. If this upper bound is less than the projected cost of maintaining an index on the domain, then this domain can safely be excluded from the initial candidate set, i.e., the domain occurs so infrequently that it can never be profitable to index it. The upper bound is calculated as follows. An index saves the largest number of page accesses if its domain only occurs in one-term queries. Under this assumption, the maximum number of page accesses is saved for each occurrence if all the qualified tuples are clustered together. Let

    p = number of pages in relation
    s = selectivity of domain
    f = number of occurrences

    then an upper bound on the number of page accesses saved is $f * p * (1 - s)$. (For each occurrence, we would have access p pages without an index, and only s*p with one).

(3) Only a small subset of all possible candidates need be considered to determine the next domain or set of domains to be adjoined to the index set at each stage. We can rank the domains by the above described upper bound and consider only the top ranking M domains, and combinations of these, for detailed incremental savings calculation. Furthermore, a bound M' ($M' \leq M$) can be put on the number of domains that will be considered together.

(4) Not all queries can use indices profitably. The expected number of tuples that will qualify for each query type can be estimated using the selectivity of the domains involved. Those that cannot profitably make use of indices are eliminated from the projected query set whose processing cost is to be minimized. This eliminates some unnecessary cost calculations. Similarly, only queries that involve domains still in the candidate set need to be retained in the query set for incremental savings calculation. (This heuristic does not cut down the search space, but does speed up the cost evaluation at each step.)

(5) An upper bound can be put on the number of cost evaluations that are performed in the entire selection procedure. The procedure is terminated when this bound is exceeded.

To illustrate the above heuristics, we present the details of our index selection procedure. Our procedure can be divided into two stages. During the first, a tentative set of domains to be indexed is chosen in an incremental fashion. It is possible to put a bound on the cost of this phase by adjusting the parameters M and M'. An alternative (or additional) constraint is to bound the total number of sets of indices for which cost evaluation is performed.

(1) Rank the domains in the relation using the procedure described above and let S be the set of domains that occur so infrequently that indexing them can not be profitable.

(2) Partition the set of domains D in the relation into three disjoint subsets: D_i - the index set, D_c - the candidate set, and D_n - the non-index set. Initialize D_i to null, D_n to S, and D_c to D - D_n.

(3) Consider in turn the incremental savings gained by indexing each of the M top ranking domains in the candidate set. Adjoin to D_i the one that will give the best improvement to the cost function. If one cannot be found, then consider larger-sized combinations (up to M') of these M domains. Consider combinations of the next larger size only if it is not profitable to adjoin any of the combinations less than or equal to the current size. Revert to consider adjoining single domains after each adjoinment. Remove the domains from D_c as they are adjoined to D_i.

(4) Terminate the first phase of the procedure if:

(a) no subset of the M (or number of domains left in the candidate set, whichever is smaller) top ranking domains up to size M' can be chosen that will improve the cost function.

(b) the upper bound on the total number of cost evaluations is reached.

The next stage may be called the bump-shift phase [12]. Domains that have been adjoined to the index set early in the first stage may turn out to be uneconomical as the result of later addition of other domains to the set, and thus should be removed from the index set. Only individual domains will be considered for removal from the tentatively chosen index set. If more than one can be chosen, then the one that leads to the best cost improvement is removed first, and the procedure repeated until a local minimum of the cost function is reached. (A possible extention to this stage is to consider the profitability of maintaining combined indices for those domains that are adjoined to the index set together.) Once an optimal set of domains for indexing has been chosen, the total cost for this choice of domains, including query processing cost, index storage, maintenance and creation (if applicable) costs, is compared with the cost for the existing set of indices. Reorganization is done only if the difference between the two is enough to cover the application program recompilation cost.

COMPARISON WITH PREVIOUS WORK

We have presented an experimental and heuristic approach to the index selection problem that is different in many respects from recent studies on index selection by Stonebraker [16], King [17], Schkolnick [11], Farley [14], and Held [15]. These other studies have either been formal analyses, which have made many simplifying assumptions in order to obtain an analytic solution, or else system designs that have been incomplete or unrealistic in various ways. Our work attempts to go farther than these by utilizing more complete and accurate models of cost and access, and by emphasizing important aspects of realistic data base environments. Below we summarize the novel aspects of our approach. We have stressed the importance of accurate usage model acquisition and data characteristic estimation in a dynamic environment where access acquirements are continually changing. We believe it necessary to apply forecasting techniques to predict future access requirements based on past observations, in order to capture and respond to the dynamic and changing nature of data base usage. Our scheme endeavours to obtain a precise model of data base usage by recording actual query patterns, thereby avoiding the strong assumption that the probabilities of two domains appearing in a query are mutually independent. We also take into consideration the facts that values of a domain are not equally likely to be used in queries, and that they are not evenly distributed among tuples of the relation, by monitoring the actual selectivities of domain values that are used in queries.

The size of actual data bases is reflected in our concern for efficient heuristics to speed up the index selection process. Our cost models account for such real overheads as the expense of index accessing and the cost of key insertion/deletion and garbage collection in index maintenance, and are based on reasonable models for data base storage. Our approach of minimizing the total processing cost for the upcoming time interval, rather than the expected cost for a single query, is flexible enough to account for the overhead costs of index creation, index storage and application program recompilation.

CONCLUSIONS

We have presented a high-level description of our approach to the problem of index selection in an adaptive data base management system that we are developing. This has been done, however, only in the restricted environment of a single-relation data base accessed through a restricted interface with limited capabilities for the selection of data. To fully realize the flexibility of a relational data base, it is necessary to consider a multi-relation environment together with a high-level non-procedural language interface that permits queries with arbitrary interconnection between relations in the qualification part and high level operators on the qualified data. In such an environment, it is necessary to consider the utility of indices for more complicated operations (such as restriction, projection, division, join, etc. [2]) and to select indices for all the relations in the data base as a whole. This is where the recording of detailed access history is necessary for optimal index selection, and the use of heuristics should be fruitful for cutting down the search space and for selecting richer index structures (such as combined indices). Our heuristic index selection procedure should be readily extendible to such an environment,

provided a cost evaluation procedure has been defined to estimate the cost of processing an arbitrary query with the presence of a particular set of indices. Our plan is to experimentally assess the optimality of our heuristic index selection algorithm on a reduced environment, before embarking on the more ambitious project of index selection for a more general environment. More fundamentally, our intent is to experimentally study the needs for, and capabilities of, a self-adaptive data management system in realistic data base environments.

ACKNOWLEDGEMENTS

This research was supported by the Advanced Research Projects Agency of the Department of Defense and was monitored by the Office of Naval Research under contract no. N00014-75-0661.

REFERENCES

[1] J. Martin, "Computer Data-Base Organization", Prentice Hall Inc., Englewood Cliffs, New Jersey, 1975.
[2] E. F. Codd, "A Relational Model of Data for Large Shared Data Banks", CACM, Vol. 13, No. 6, June, 1970.
[3] J. B. Rothnie, T. Lozano, "Attribute Based File Organization in a Paged Memory Environment", CACM, Vol. 17, No. 2, Feb., 1974.
[4] P. C. Yue, C. K. Wong, "Storage Cost Considerations in Secondary Index Selection", International Journal of Computer and Information Sciences, Vol. 4, No. 4, 1975.
[5] A. F. Cardenas, "Analysis and Performance of Inverted Data Base Structures", CACM, Vol. 18, No. 5, May, 1975.
[6] R. Bayer, E. McCreight, "Organization and Maintenance of Large Ordered Indexes", Acta Informatica, Vol. 1, Fasc. 3, 1972.
[7] M. M. Astrahan, D. D. Chamberlin, "Implementation of a Structured English Query Language", Proceedings of the ACM-SIGMOD Conference on the Management of Data, May, 1975.
[8] C. L. Liu, "Introduction to Combinatorial Mathematics" McGraw-Hill Book Company, 1968.
[9] R. G. Brown, "Statistical Forecasting for Inventory Control", McGraw-Hill Book Company, 1959.
[10] R. G. Brown, "Smoothing, Forecasting and Prediction of Discrete Time Series", Prentice Hall Inc., Englewood Cliffs, New Jersey, 1962.
[11] M. Schkolnick, "The Optimal Selection of Secondary Indices For Files", Research Report, Department of Computer Science, Carnegie-Mellon University, Nov., 1974.
[12] A. A. Kuehn, M. J. Hamburger, "A Heuristic Program for Locating Warehouses", Management Science, Vol. 9, No. 4, July, 1963.
[13] R. C. Meier, W. T. Newell, H. L. Pazer, "Simulation in Business and Economics", Prentice Hall Inc., Englewood Cliffs, New Jersey, 1963.
[14] J. H. G. Farley, S. A. Schuster, "Query Execution and Index Selection for Relational Data Bases", Technical Report CSRG-53, University of Toronto, Mar., 1975.
[15] G. D. Held, "Storage Structures for Relational Data Base Management Systems", Memorandum No. ERL-M533, University of California, Berkeley, Aug., 1975.
[16] M. Stonebraker, "The Choice of Partial Inversions and Combined Indices", International Journal of Computer and Information Sciences, Vol. 3, No. 2, 1974.
[17] W. F. King, "On the Selection of Indices for a File", IBM Research RJ 1341, San Jose, Jan., 1974.