Self-adaptive automatic data base design*

by MICHAEL HAMMER
MIT Laboratory for Computer Science
Cambridge, Massachusetts

ABSTRACT

Physical data base design, the selection of organizational structures and access mechanisms for a data base, is one of the most important responsibilities of a Data Base Administrator (DBA). A DBA often has difficulty in performing this task; he lacks the information needed to choose a design that is well matched to the data base's mode of use.

This paper presents the design principles of an automatic system that has the ability to choose the physical design for a data base and to adapt this design to changing requirements. The components of such a system include: an information gathering module that collects global statistics on the overall usage pattern of the data base; a predictor that projects observed usage statistics into the future; a design evaluator that computes a figure of merit for any proposed design; and a heuristic proposer that synthesizes a small set of candidate designs for detailed consideration. These principles have been applied to the design of a system that selects secondary indices for an inverted file system.

INTRODUCTION

The advent of shared integrated data bases has called into being a new job function, that of the Data Base Administrator (DBA). The DBA is charged with responsibility for a collection of data that is being used in differing ways, for varying reasons, by a disparate community of users; he has authority over an important resource that no single user can (or should) control. It is his task to mediate the conflicting needs of the user community and make decisions regarding the organization and maintenance of the data base. The particular tasks associated with this important position cannot be easily summarized in a job description. By all accounts, one of the DBA's major areas of responsibility is for the physical design of the data base; by this, we mean all decisions pertaining to the structure and representation of the data and its associated access mechanisms.

The physical design of a data base will determine the cost of its use and maintenance: the time it takes to perform retrievals and updates, and the space needed to store the data and its associated auxiliary structures. (A terminological note: We use "update" as a generic term to include insertions and deletions of records as well as the modification of existing records.) In an important sense, physical design issues should be largely invisible to users of the data base: i.e., they should primarily affect only system performance and not the ways that users view the data or write their programs. The latter sort of concerns are usually gathered together under the term "logical design." In some contemporary systems, the choice of a logical schema does have a direct impact on the structure of the physical representation of the data, and so performance issues do impact the users' view of the data. But with the growing trend towards data independence in modern systems, the logical and physical design processes are being increasingly decoupled. Logical design is an exercise in modelling, wherein the DBA attempts to design a logical data structure that effectively captures the fundamental semantics of the application domain and thus enables users to express their transactions with the data base in a natural and convenient way. Physical design can be viewed as the process of reducing this abstract representation to a concrete one, which is expressed in terms of physical data structures and access mechanisms; the only relevant issue at this level should be performance.

Physical design encompasses an extensive set of issues: which ones are relevant in the design of a particular data base depends on the kinds of file structures and access methods that are supported by the data base management system (DBMS) being used. Typical design problems in this domain include: choosing the primary access methods for the files of a data base (e.g., sequential, indexed sequential, or direct); for a sequential file, selecting the major and minor keys by which the file is to be sorted; identifying the set of fields for which to maintain indices, and choosing the structures of these indices.

In most cases, the set of possible physical designs for a given logical data base is very large. No single one of them is the optimal in all circumstances. Rather, one design can be said to be better or worse than another only in the context of a particular pattern of use for the data base. It is well-known that any particular physical design of a data base will enable the efficient execution of certain retrievals but not of others, and that it will entail extensive mainte-
nance activity for certain updating activities but not for others. Thus, the objective of the physical design process is the selection of a physical representation that will provide good performance in the context of the particular mix of retrievals and updates to which the data base will be subjected.

There is another factor that affects the choice of the physical design for the data base, which might be called the internal characteristics of the data. This includes such issues as the sizes of files, the ranges of values that fields can assume, the distributions of values in fields across a record type, the cardinalities of relationships between record types, the typical numbers of elements in repeating groups; these all affect how a particular physical design will perform for a set of transactions, and so should influence the choice of the physical design.

Large, shared data bases are not static entities. When a data base is used as an autonomous organizational resource, rather than as a set of master files for some particular programs, it develops a highly dynamic life cycle of its own. Its usage patterns can shift dramatically as old applications evolve or are replaced, as new applications emerge, as users acquire increased sophistication and familiarity with the system. Internal characteristics of the data may change as well, reflecting the changing nature of the domain being modelled by the data base. Consequently, the tuning of a data base's physical organization must also be a continual process; physical redesign, to meet changing requirements, is as important as the initial configuration process.

In light of all these issues that influence the choice of a good design, it is not surprising that a physical design chosen by the DBA in an intuitive fashion, based on his qualitative impressions, is likely to result in unsatisfactory performance. But problems also face the DBA in trying to choose a design in a more systematic fashion. First of all, it is difficult for him to obtain an accurate usage pattern for the data base as a whole, since he typically has meaningful interactions only with a limited number of individual users. Secondly, a modern data management system is a complex device, and it is hard for a human DBA to develop a useful intuitive model of its operation; on the other hand, a more precise model would be so complex that the evaluation of any proposed design would be an extremely cumbersome process. Next, even if the DBA were able to assign a figure of merit to any particular design, he would not be able to use this ability to select an optimal structure, because the number of candidate organizations is almost certain to be too large to allow their individual consideration. Finally, because of the evolving character of data base usage and data characteristics, even a well-chosen design is not likely to remain a good one for long. Obtaining meaningful predictions as to what the relevant parameters affecting the design will be like in the future is even harder than determining what they have been in the past.

Therefore, the DBA needs help in choosing the physical design for his data base; and it is natural to look to the data management system itself to assist in (or even subsume) this process. It certainly is the best source for all the information needed to do the task effectively. Since the DBMS processes all user transactions it has the capability to build up an integrated and accurate model of data base usage; it should also be able to detect emerging trends and project them into the future. Similarly, it can obtain summaries of the pertinent internal characteristics of the data during the normal course of its operation. It could also be provided with a detailed model of its own operation, in order to compute useful cost figures for candidate physical designs. What we are proposing is imbuing the data management system with a limited form of "intelligence", thereby enabling it to share in the making of decisions heretofore entirely within the province of humans.

In the remainder of this paper, we describe how a data management system can participate in the physical design process; initially, by means of a set of tools to aid a human DBA, and then through a totally automated data base design facility.

INFORMATION GATHERING AND DESIGN EVALUATION

The basic components of any automated or semi-automated physical design system are a design evaluation module and an information gathering module. The former is used to comparatively evaluate any set of candidate designs for the data base and thereby rationalize the process by which a design is selected. In order to perform such automatic physical design evaluation, the system must have access to information that characterizes the use of the data base and its contents. It is the responsibility of the information gathering module to collect such data. This module can be incorporated into the interface language processor of the DBMS; it can be thought of as watching over the shoulder of the language processor and taking notes on what it sees. The choice of what information it should collect is determined by the design issues to be addressed and the approach being taken to design evaluation by the evaluator module. This choice should also be influenced by considerations of accuracy and efficiency. On the one hand, succinct summary statistics may not provide enough data for the precise analysis of possible designs. On the other, it may be prohibitively expensive to gather and store a large amount of data of a fine-grained character; furthermore, it might be difficult to make effective use of a mass of low-level facts.

Earlier efforts in the area of automatic data base design have utilized rough summary information to describe the use of a data base and its contents. This description of a usage pattern in terms of a small number of parameters has been used to characterize an "average" transaction, with respect to which any proposed design could be benchmarked. The difficulty with this approach is that it obscures a great deal of detail that is crucial in tuning a data base design to match its mode of use; much relevant information is lost in describing a usage pattern in terms of a single average transaction.

We believe that for most design decisions, the most
effective way to evaluate a proposed design is by examining its aggregate performance for all the individual transactions that comprise the usage of the data base. The system should utilize a model of the operation of the DBMS, together with information about the internal characteristics of the data, to forecast how much processing would be done by the DBMS in the handling of each individual transaction; the sum of these processing costs over all transactions in a usage pattern can be combined with certain other cost figures to achieve an overall figure of merit for the proposed design. Therefore we shall attempt to gather as much detailed information as possible about the individual transactions with the data base and its internal characteristics.

Practically, it would be infeasible to maintain a separate record of every transaction that occurs with the data base; it would also be prohibitively expensive to attempt to analyze each one when considering a proposed design. Therefore, we shall partition the set of transactions into transaction classes such that all transactions in the same class will have the same essential structure. For example, the class of a retrieval request will be determined by the field names and comparison operators occurring in the atomic predicates and by the structure of the total predicate in terms of its logical connectives. The intent is that all transactions in the same class should entail roughly the same amount of processing by the DBMS; while this may not be completely accurate, we believe it represents an acceptable tradeoff between efficiency and accuracy. Then a usage pattern will be expressed by the occurrence frequencies of the various transaction classes; a design will be evaluated by considering its (weighted) performance for the different classes, as predicted by the model of the DBMS. Henceforth, we shall take "transaction" to mean "transaction class."

There are two further points to be observed about the information gathering module. First, it is imperative that the process of gathering information not significantly degrade DBMS performance; the statistics that this module collects ought to be readily available in the normal operation of the DBMS. Secondly, the usage pattern may have to reflect more than just the occurrence frequencies of the various transactions. It may well be that certain users or applications have a higher priority than others, and that their needs and habits should carry more weight in determining the physical design of the data base.

The information gathering module provides the background knowledge essential to choosing a physical design; this information is utilized by the physical design evaluator. This module takes two inputs: the output of the information gatherer (a description of the usage pattern and internal characteristics of the data bases) and a proposed physical design for the data base. Its function is to produce a figure of merit that reflects the suitability of the proposed physical design for a data base with the specified internal characteristics that is being used in the way described by the usage pattern.

The evaluator will be used to compare alternative designs and select the best of any set of candidates. The figure of merit need not be a completely accurate measure of the cost associated with an individual design, but it must allow for reliable comparisons to be made of alternative designs.

The design evaluator will be built around a transaction cost estimator. This estimator is called with a transaction, a proposed physical design for a data base, and a summary of the data base's internal characteristics; it computes a figure that reflects the cost that the DBMS would incur in processing the transaction, if the data base were structured in the proposed way. The units of this estimate could be expected I/O processing, or expected CPU time, or a combination of the two.

The bulk of a (naive) design evaluator could consist, then, simply of an iteration over the set of transactions in the usage pattern; each transaction would be submitted to the cost estimator, and the result tallied into a running total. The value computed by this iteration could be combined with a figure reflecting the costs of storing and maintaining the access structures, in order to achieve a unified figure of merit for the proposed design. We shall discuss later the shortcomings of this naive evaluator and how they may be repaired; however, in the interim, this can serve as a useful model of the structure of this module.

The transaction cost estimator requires a detailed model of the operation of the DBMS. By scanning the structure of a transaction, this module will determine what strategy the DBMS would follow in processing it: what access mechanisms would be utilized and in what order, what kinds of intermediate results would be generated, and so on. The detailed information about the data base's internal characteristics will enable the estimator to forecast such things as the sizes of the various structures the DBMS would have to scan and the number of links it would have to traverse. Thus the estimator will be able to predict the total amount of activity required of the DBMS to handle this transaction. Since the estimator will have to operate efficiently, it may have to ignore some hard-to-compute factors in the processing cost; but the dominant terms should be readily approximated.

Such a cost estimator is an entity of interest in its own right, independent of the problem of self-organizing data bases. Many contemporary data base systems provide a "stand-alone" language interface, which is intended for use by relatively unsophisticated users in posing ad hoc inquiries to the data base in real-time. An unfortunate property of such a facility is that it enables a naive user to ask the system a seemingly simple query whose processing will cost a great deal more (either in terms of elapsed time or other system resources) than the answer is worth to him. A transaction cost estimator could inspect any query and quickly return to the user an estimate of its processing cost, enabling him to abort expensive queries and to better plan his data base usage.

Although the information gatherer and the design evaluator form a coherent unit, either one by itself would still be useful to the DBA: the information gatherer could provide him with data for his use in intuitively designing file structures, or the design evaluator could operate with usage statistics that he supplied it.
Theoretically, the DBA could run the evaluator on the full set of all possible designs and select the one with the best figure of merit; he would then be reasonably certain of having chosen a near-optimal structure. More realistically, the DBA would intuitively choose some small set of structures to investigate more closely, and subject them to careful analysis and evaluation.

REDESIGN AND USAGE PREDICTION

A scenario of the use of these basic DBA design tools is as follows. At the time of data base creation, the DBA generates a pattern that he believes represents the expected usage pattern of the data base, and uses it as input to the evaluator in selecting the initial physical design. He then activates the information gatherer to monitor the actual transactions with the data base and determine its real mode of use. When sufficient statistics have been gathered, the DBA can decide if his initial guess was accurate. If not, he can use the evaluator again and choose another design that better fits the observed usage.

The flaw in this primitive scenario is, of course, that a data base usage pattern is not static, and so even a well-chosen design may become obsolete. Consequently, it is desirable to incorporate the concept of redesign into this picture. To accomplish this, we introduce the notion of redesign points, which could occur on a regular basis or on DBA request (for example, upon his noticing a significant degradation in system performance). At each such point, the statistics gathered since the last redesign point are summarized into a usage pattern, which presumably defines the most recent mode of use. The evaluator can then be used to help the DBA select a physical design optimally suited to this usage pattern. This design is the structure of choice in the current circumstances.

This approach raises some issues that must be addressed. One of these involves the cost of performing a redesign. If it develops that the optimal design for the evidenced usage pattern is different from the existing design, then there is almost certainly going to be some cost associated with the process of transforming the data base from the old physical organization into the new one. The extent of this cost depends on the size of the data base, the kinds of structures involved, and the degree of the difference between the old and new designs. It may be the case that the cost improvement to be gained by using the new design rather than the old will be washed out by the expense of the reconfiguration process. Consequently, this latter cost must also be considered in choosing the physical design. The optimal design is thus defined as that design D for which F(D)=C(D)−C(Do)−T(Do,D) is a maximum, where Do is the existing design, C(D) is the total cost associated with design D for the usage pattern observed since the last design point (as computed by the evaluator), and T(Do,D) is the cost of transforming the data base from design Do to design D. The evaluator can readily be modified to compute F(D) for any proposed design.

The other problem with our redesign scenario is rather more fundamental. In the process described above, we are always designing for the past, optimizing for a usage pattern that is already over. The tacit assumption behind this is that the usage of the future will strongly resemble that of the past. However, it may happen that the redesign points are badly situated with respect to the evolution of the usage pattern, and that statistics gathered since the last design point are dominated by transactions least representative of future usage. This situation might well obtain if redesign points occur when system performance is just beginning to degrade.

For this reason, it is appropriate to be more sensitive to the problem of changing usage patterns and to include in the system a predictor module. This component will interpret the observed statistics and explicitly translate them into an anticipated future usage pattern. The intent is that this module will detect evolving trends in data base use even before they become fully established, and so enable the data base structure to be prepared in advance for the demands to be put on it. A predictor module needs access not only to the usage statistics of the most recent time interval, but to those of earlier periods as well. By analyzing the historical trends of the various parameters that comprise the usage pattern, this component can synthesize a description of expected use for the upcoming interval. It is with respect to this predicted pattern that the alternative physical designs will be evaluated.

A good predictor module must satisfy a number of criteria. It must not be overly vulnerable to chance fluctuations in usage, while still responding to real change. Therefore, it cannot base its decisions purely on recent events, nor on the aggregate of information gathered over the entire history of the data base. Intuitively, some weighted average of the two is desirable. At the same time, the predictor cannot demand access to a great deal of detailed information from all previous time periods, since both the gathering and storage of such information is likely to be prohibitive in cost; nor can it attempt to perform extensive computations for each prediction that it makes. A useful usage pattern is likely to be composed of a large number of parameters, and a lengthy evaluation for each one is impractical.

A promising predictive technique for this application is exponential smoothing. The basic formulation of exponential smoothing in making a forecast of a discrete time series is as follows:

\[
\text{new forecast}=a*(\text{actual observation from last period})+(1-a)*\text{(previous forecast)}
\]

where a is called a smoothing constant and takes on values between 0 and 1. In essence, this computes 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 the smoothing constant a can be selected by the
DBA or can be adaptively chosen by the system itself to minimize the difference between observed and predicted data.) Presumably, the predictor would maintain a time series for each parameter in the usage pattern. In this scheme, only two values need be maintained for each series: the current observation and the previous forecast. The computational requirements of this approach are also minimal.

This basic scheme can be modified in order to make it more sensitive to evolving trends. The revised formulation is as follows:

\[
\begin{align*}
\text{new average} &= a \times (\text{current observation}) + (1-a) \times (\text{old average}) \\
\text{current trend} &= \text{new average} - \text{old average} \\
\text{new trend} &= a \times (\text{current trend}) + (1-a) \times (\text{old trend}) \\
\text{new forecast} &= \text{new average} + (1-a) \times (\text{new trend})
\end{align*}
\]

The new trend is a smoothed average of the differences between successive basic forecasts (as formulated above), and so represents the direction that these forecasts are taking. The revised forecast is the basic forecast modified by a weighted version of the new trend. Here, too, storage requirements are minimal; space is needed only for the current observation, the old average, and the old trend.

A number of issues must be confronted in attempting to apply exponential smoothing to data base usage prediction. One question is whether such techniques, developed for problem domains like inventory control, are really appropriate for the data base environment. The fundamental problem is determining the nature of the patterns and trends that do occur in the history of use of a centralized data base by a diverse community of users. Certainly, such histories seem to have some aspects that are not congenial to modelling by smoothing techniques. Ultimately the validity of this approach to usage prediction can only be determined by careful study of extensive amounts of appropriate data that have been gathered in an operational environment.

AN AUTOMATIC DESIGN FACILITY

As we have described it, the information gathering module collects the statistics on observed data base use, and passes them to the predictor module, where they are converted into an expected usage pattern. The design evaluator utilizes this prediction in assessing possible designs. What, then, remains the function of the human DBA (with respect to the data base design problem)? It might be said that his first responsibility in this area is that of any human working with an automated decision-making system: to assure that the system’s decisions are sensible, that they reflect not only an abstract model but concrete reality as well. Thus the DBA should interpret the statistics that are gathered and the usage that is forecasted, decide if they are plausible, and manually modify them if he feels they are incomplete. Similarly, he must exercise human judgment in his use of the design evaluator, deciding what its results mean in terms of actual data base design. The DBA may need to go beyond the limited contexts of these automated tools, in order to account for hard-to-quantify factors not incorporated in their computations and consequently not reflected in their output. For example, the DBA might decide that the best design is not the one with the lowest cost figure as computed by the evaluator; his decision may be influenced by taste and intuition.

There is a problem with the foregoing argument: while the kind of judgment described above is fundamentally beyond the capacity of a machine, it may also effectively exceed a human’s capability as well. With large data bases being used in a wide variety of ways, and with complex data management systems that make clever use of intricate storage structures, a DBA may find that he does not have the subconscious understanding of “what’s really going on” needed to transcend the purely quantitative capabilities of an automated design system. He may be forced to accept the decisions of his design aids, because he does not have any real basis for overruling them.

Consequently, if the DBA is provided with an information gatherer, a predictor, and a design evaluator, his role in the physical design process is reduced to deciding which physical designs warrant detailed examination by the evaluator. To be sure, synthesizing a good set of candidate designs is a creative task and an extremely important one. In modern data base systems, the physical design space may have many dimensions, with a large number of alternatives on each axis. The total number of possible organizations for even a simple data base is likely to be astronomical; an exhaustive iteration over all of them is clearly impossible. Thus, human intervention seems necessary in selecting a small set of promising candidates to submit to detailed evaluation.

But here too, a human's performance may be less than acceptable, for the reasons described above; the DBA may not be able to identify any of the near-optimal designs, and may send to the evaluator a set containing only mediocre alternatives. Consequently, we arrive at the notion of a completely automated designer. Previous attempts at automated physical design have attempted to derive a closed-form analytic expression for the cost associated with a design that is parameterized in terms of key properties of the design; this expression is subjected to mathematical optimization techniques in order to yield an optimal design. It is our belief that reducing complex designs to tractable formulae inevitably entails their serious oversimplification. Our approach to automatic design relies on an intelligent module to pick a small set of promising design candidates, which can then be sent to the evaluator for detailed analysis.

Therefore, our automatic design system is comprised of the modules described above, plus a design proposer. At each design point, the proposer will inspect the predicted usage pattern and propose a set of designs to be analyzed by the evaluator. Of course, a trivial proposer could propose all possible designs, but that is clearly unrealistic. Therefore, the proposer must employ heuristics to choose a manageable number of candidate designs. There is the possibility that in some cases such heuristics will not
synthesize the mathematically optimal solution; but if the heuristics are well chosen, then in all but the most pathological of contexts, the best design that they do produce should perform nearly as well as the optimal. The synthesis of a very good (if not optimal) design, at low cost, is certainly an acceptable goal for an automatic designer. The competence of a heuristic designer can be verified through experimentation, by comparing its designs in a variety of test cases to those that would be chosen after an exhaustive consideration of all designs.

It is useful to view the problem of automatic physical design as essentially one of navigating through a large search space (that of all possible designs) while looking for the optimal point. This suggests that heuristic search techniques developed for artificial intelligence applications should prove useful in selecting a near-optimal design. Some of these techniques include the following: incremental search through the problem space, with no backtracking and transition made only to improved positions; early termination of the search when a local optimum is reached; decoupling of related design decisions, with a relative ordering placed on them; a rough a priori ranking on alternatives in each design dimension, with primary attention to be paid to high-ranking alternatives.

The foregoing suggests a system design wherein the proposer operates with feedback from the evaluator. That is, the proposer begins with some initial candidate design (which is either a trivial one, or one chosen on the basis of obvious surface properties of the usage pattern, or the actual current design of the data base). The proposer then constructs a small set of variations on this candidate, based on some a priori judgments. These variations are then sent to the evaluator for analysis. That variation which represents the greatest total cost improvement over the current candidate becomes the new candidate design. (If none of its variations represents an improvement, another small set of alternatives may be considered.) The information gained in evaluating these variations determines which variations of the new candidate ought to be considered at the next step. This process continues until a local optimum is reached: a design which is an improvement over its predecessor, but over which none of the considered variations yields a better figure from the evaluator.

In cases of multiple design decisions, the foregoing procedure would be applied sequentially in the different dimensions, based on some general guidelines as to their relative importance.

Experimentation with the heuristics can determine the values of the operating parameters that achieve an acceptable balance between accuracy and efficiency. For example, by limiting the numbers of alternatives that are considered at each step, the algorithm can severely restrict the number of designs that are submitted to the evaluator.

The efficiency ofthe designer can be further enhanced by modifying the structure of the design evaluator. The basic difficulty with the evaluator as initially described is that it applies the transaction cost estimator to each component of the projected usage pattern; this might be expensive if many of the possible transactions with the data base are expected to occur. One approach to this problem is based on the observation that, when evaluating a design that is a minor variation of the current candidate design, most transactions will have the same cost as they did when the current candidate was evaluated; the reason for this is that most transactions will be processed in exactly the same way for two similar physical designs. Since we are only interested in the comparative evaluations of the candidate and the proposed variation, the evaluator can concentrate on those transactions that have different costs in the two cases. Another approach is to group together similar transactions into a cluster to be represented by a typical transaction; the estimator is then applied to just the representative, and the result multiplied by the size of the cluster. (This is an extension of the concept of grouping transactions into transaction classes.) This can result in a certain inaccuracy, because not all transactions in a cluster will have the same cost. This scheme is probably most effective when infrequently-occurring transactions are clustered together, while the more common ones are still treated individually. The tradeoff between efficiency and accuracy is controlled by the total number of clusters; a good value for this parameter can be obtained through experimentation.

A self-organizing data base system should be able to decide when, as well as how, to redesign a data base to fit changing access requirements; the redesigner ought to be automatically invoked when the current design begins to show signs of degraded performance. Two conditions have to be satisfied in order to signal this situation. First, that the recent pattern of transactions is deviating sharply from the predicted usage pattern to which the current design is tuned. Second, that system performance in the aggregate is proving to be less good than it had been in the past. Some caution must be exercised in invoking the redesigner, so that a transient fluctuation in usage does not cause the (somewhat expensive) process of considering a redesign to be activated. On the other hand, it is too late to wait for degraded performance to be firmly established before beginning the redesign process.

A SYSTEM FOR SECONDARY INDEX SELECTION

The foregoing discussion has been couched in general terms; the system structure described should be applicable to a wide variety of physical design decisions pertinent in different data management systems. We are developing a facility that applies these principles to the problem of selecting secondary indices for data bases managed by an inverted file data management system. A secondary index for a field A is a structure that, for a given value x, provides rapid access to the identifiers (addresses) of all records in the file whose value for A is x. A major design decision for a data base under a system that supports secondary indices is the selection of the appropriate set of fields for which indices ought to be maintained. An index on field A will speed up retrievals involving A, but will slow down updates to A (as well as requiring extra storage space). The accurate determination of whether or not a particular field ought to be indexed depends on a range of considerations beyond its
relative occurrence frequencies in retrievals and updates. These include: the selectivity of the field, or the extent to which its presence in a retrieval request cuts down the number of records satisfying the request and so affects the time needed for its processing; the kinds of retrieval predicates in which the field is used; which other fields are known to be indexed; and the details of the DBMS strategies for processing transactions.

Our facility selects indices for an experimental data management system similar in structure to several commercial inverted file systems. The details of this facility are presented elsewhere [6,7]; some of its major points of interest follow.

1. The data management system operates in a paged virtual memory operating system. Since I/O processing is usually the dominant cost in data base systems, the transaction cost estimator computes the expected number of page accesses associated with the processing of the transaction. The overall figure of merit computed by the physical design evaluator is also expressed in units of page accesses; other kinds of costs (e.g., for storage of indices) are converted into this scale.

2. The major information needed by the transaction cost estimator in determining the cost of processing a retrieval is the average selectivity of each field that is used in the predicate and that is indexed in the proposed design. This information can readily be obtained by observing the intermediate results computed by the transaction processor while handling retrieval requests.

3. The relevant usage pattern information is easy to capture during transaction processing.

4. The techniques underlying the cost estimator have been applied to an operational inverted file system, with encouraging results. Using appropriate selectivity information, estimates are produced that approximate very closely the number of page accesses actually performed by the system in processing transactions.

5. The design proposer employs a set of heuristics that enable it to send a relatively small set of candidate designs to the evaluator for analysis. This heuristic proposer has been tested against an exhaustive proposer for a wide range of usage patterns and data characteristics. In virtually all cases, the heuristic proposer finds the optimal design; in the remaining instances, its selection has a total cost within five percent of the true optimum. The number of evaluator calls that the heuristic proposer makes is roughly proportional to the number of fields in the file, compared to an exponential number for the exhaustive version.

SUMMARY

We have presented the principles of a novel approach to the design of self-organizing, adaptive data base systems. This approach differs from earlier efforts in the field by its use of detailed information on data base usage, its concern with evolving trends in the usage pattern, its transaction by transaction analysis of every proposed design, and its reliance on heuristics to synthesize a small set of candidate designs. The keystone of a system built on these principles is a transaction cost estimator, which assesses the cost of performing any specified transaction in the context of a proposed design. An information gathering module acquires sufficient knowledge about the contents of the data base to enable the cost estimator to operate; it also records the global usage pattern of the data base. At each redesign point, all this information is projected into the future, so that the design should be matched to developing requirements. The design evaluator combines the cost figures associated with the transactions in the usage pattern with general maintenance and storage costs to achieve a unified figure of merit for the design. Both the design proposer and the design evaluator make use of heuristics: the former to guide its search through the space of potential designs and thereby restrict the number of calls on the evaluator, and the latter to coalesce transaction classes and reduce the number of calls on the estimator.

These principles have been applied to the design of a system that selects secondary indices for an inverted file DBMS. A transaction cost estimator has been implemented that is efficient and also quite accurate in its forecasts of page accesses; the information that it requires can readily be gathered during normal transaction processing; and its associated heuristic-based proposer performs virtually as well as its exhaustive counterpart, but at dramatically reduced cost. This facility is being extended to address such issues as the selection of sort keys and file partitioning.

REFERENCES
