1

Online Index Recommendations for High-Dimensional Databases using Query Workloads

Michael Gibas, Guadalupe Canahuate, Hakan Ferhatosmanoglu

***Abstract*— High-dimensional databases pose a challenge with respect to efficient access. High-dimensional indexes do not work because of the oft-cited 'curse of dimensionality'. However, users are usually interested in querying data over a relatively small subset of the entire attribute set at a time. A potential solution is to use lower dimensional indexes that accurately represent the user access patterns. Query response using physical database design developed based on a static snapshot of the query workload may significantly degrade if the query patterns change. To address these issues, we introduce a parameterizable technique to recommend indexes based on index types frequently used for high-dimensional data sets and to dynamically adjust indexes as the underlying query workload changes. We incorporate a query pattern change detection mechanism to determine when the access patterns have changed enough to warrant change in the physical database design. By adjusting analysis parameters, we trade off analysis speed against analysis resolution. We perform experiments with a number of data sets, query sets, and parameters to show the effect that varying these characteristics has on analysis results.**

***Index Terms*— index selection, high-dimensional indexing, query access patterns**

I. Introduction

An increasing number of database applications, such as business data warehouses and scientific data repositories, deal with high dimensional data sets. As the number of dimensions/attributes and overall size of data sets increase, it becomes essential to efficiently retrieve specific queried data from the database in order to effectively utilize the database. Indexing support is needed to effectively prune out significant portions of the data set that are not relevant for the queries. Multi-dimensional indexing, dimensionality reduction, and RDBMS index selection tools all could be applied to the problem. However, for high-dimensional data sets, each of these potential solutions has inherent problems.

To illustrate these problems, consider a uniformly distributed data set of 1,000,000 data objects with several hundred attributes. Range queries are consistently executed over five of the attributes. The query selectivity over each attribute is 0.1, so the overall query selectivity is $1/10^5$ (i.e. the answer set contains about 10 results). An ideal solution would allow us to read from disk only those pages that contain matching answers to the query. We could build a multi-dimensional index over the data set so that we can directly answer any query by only using the index. However, performance of multi-dimensional index structures is subject to Bellman's curse of dimensionality [1] and degrades rapidly as the number of dimensions increases. For the given example, such an index would perform much worse than sequential scan. Another possibility would be to build an index over each single dimension. The effectiveness of this approach is limited to the amount of search space that can be pruned by a single dimension (in the example the search space would only be pruned to 100,000 objects).

For data-partitioning indexes such as the R-tree family of indexes, data is placed in a partition that contains the data point and could overlap with other partitions. To answer a query, all potentially matching search paths must be explored. As the dimensionality of the index increases, the overlaps between partitions increase and at high enough dimensions the entire data space needs to be explored. For space-partitioning structures, where partitions do not overlap and data points are associated with cells which contain them (e.g. grid files), the problem is the exponential explosion of the number of cells. A 100 dimensional index with only a single split per attribute results in 2^{100} cells. The data to cell ratio can be small enough that the cells can not be efficiently searched. This phenomenon is thoroughly described in [2].

Another possible solution would be to use some dimensionality reduction technique, index the reduced dimension data space, and transform the query in the same way that the data was transformed. However, the dimensionality reduction approaches are mostly based on data statistics, and perform poorly especially when the data is not highly correlated. They also introduce a significant overhead in the processing of queries.

Another possible solution is to apply feature selection to keep the most important attributes of the data according to some criteria and index the reduced dimensionality space. However, traditional feature selection techniques are based on selecting attributes that yield the best classification capabilities. Therefore, they also select attributes based on data statistics to support classification accuracy rather than focusing on query performance and workload in a database domain. As well, the selected features may offer little or no data pruning capability given query attributes.

Many commercial RDBMS's have included index recommendation systems to identify indexes that will work well for a given workload. These tools are optimized for the domains for which these systems are primarily employed and the indexes that the systems provide. They are targeted towards lower dimension transactional databases and do not produce results optimized for single high-dimensional tables.

Our approach is based on the observation that in many high dimensional database applications, only a small subset of the overall data dimensions are popular for a majority of queries and recurring patterns of dimensions queried occur. For example, Large Hadron Collider (LHC) experiments are expected to generate data with up to 500 attributes at the rate of 20 to 40 per second [3]. However, the search criterion is expected to consist

The authors are with The Ohio State University

of 10 to 30 parameters. Another example is High Energy Physics (HEP) experiments [4] where sub-atomic particles are accelerated to nearly the speed of light, forcing their collision. Each such collision generates on the order of 1-10MBs of raw data, which corresponds to 300TBs of data per year consisting of 100-500 million objects. The queries are predominantly range queries and involve mostly around 5 dimensions out of a total of 200.

We address the high dimensional database indexing problem by selecting a set of lower dimensional indexes based on joint consideration of query patterns and data statistics. This approach is also analogous to dimensionality reduction or feature selection with the novelty that the reduction is specifically designed for reducing query response times, rather than maintaining data energy as is the case for traditional approaches. Our reduction considers both data and access patterns, and results in multiple and potentially overlapping sets of dimensions, rather than a single set. The new set of low dimensional indexes is designed to address a large portion of expected queries and allow effective pruning of the data space to answer those queries.

Query pattern evolution over time presents another challenging problem. Researchers have proposed workload based index recommendation techniques. Their long term effectiveness is dependent on the stability of the query workload. However, query access patterns may change over time becoming completely dissimilar from the patterns on which the index set were originally determined. There are many common reasons why query patterns change. Pattern change could be the result of periodic time variation (e.g. different database uses at different times of the month or day), a change in the focus of user knowledge discovery (e.g. a researcher discovery spawns new query patterns), a change in the popularity of a search attribute (e.g. current events cause an increase in queries for certain search attributes), or simply random variation of query attributes. When the current query patterns are substantially different from the query patterns used to recommend the database indexes, the system performance will degrade drastically since incoming queries do not benefit from the existing indexes. To make this approach practical in the presence of query pattern change, the index set should evolve with the query patterns. For this reason, we introduce a dynamic mechanism to detect when the access patterns have changed enough that either the introduction of a new index, the replacement of an existing index, or the construction of an entirely new index set is beneficial.

Because of the need to proactively monitor query patterns and query performance quickly, the index selection technique we have developed uses an abstract representation of the query workload and the data set that can be adjusted to yield faster analysis. We generate this abstract representation of the query workload by mining patterns in the workload. The query workload representation consists of a set of attribute sets that occur frequently over the entire query set that have non-empty intersections with the attributes of the query, for each query. To estimate the query cost, the data set is represented by a multi-dimensional histogram where each unique value represents an approximation of data and contains a count of the number of records that match that approximation. For each possible index for each query, the estimated cost of using that index for the query is computed.

Initial index selection occurs by traversing the query workload representation and determining which frequently occurring attribute set results in the greatest benefit over the entire query set. This process is iterated until some indexing constraint is met or no further improvement is achieved by adding additional indexes. Analysis speed and granularity is affected by tuning the resolution of the abstract representations. The number of potential indexes considered is affected by adjusting data mining support level. The size of the multi-dimensional histogram affects the accuracy of the cost estimates associated with using an index for a query.

In order to facilitate online index selection, we propose a control feedback system with two loops, a fine grain control loop and a coarse control loop. As new queries arrive, we monitor the ratio of potential performance to actual performance of the system in terms of cost and based on the parameters set for the control feedback loops, we make major or minor changes to the recommended index set.

The contributions of this paper can be summarized as follows:

1) Introduction of a flexible index selection technique designed for high-dimensional data sets that uses an abstract representation of the data set and query workload. The resolution of the abstract representation can be tuned to achieve either a high ratio of index-covered queries for static index selection or fast index selection to facilitate online index selection.
2) Introduction of a technique using control feedback to monitor when online query access patterns change and to recommend index set changes for high-dimensional data sets.
3) Presentation of a novel data quantization technique optimized for query workloads.
4) Experimental analysis showing the effects of varying abstract representation parameters on static and online index selection performance and showing the effects of varying control feedback parameters on change detection response.

The rest of the paper is organized as follows. Section II presents the related work in this area. Section III explains our proposed index selection and control feedback framework. Section IV presents the empirical analysis. We conclude in Section V.

II. Related Work

High-dimensional indexing, feature selection, and DBMS index selection tools are possible alternatives for addressing the problem of answering queries over a subspace of high-dimensional data sets. As described below, each of these methods provides less than ideal solutions for the problem of fast high-dimensional data access. Our work differs from the related index selection work in that we provide a index selection framework that can be tuned for speed or accuracy. Our technique is optimized to take advantage of multi-dimensional pruning offered by multi-dimensional index structures. It takes into consideration both data and query characteristics and can be applied to perform real-time index recommendations for evolving query patterns.

A. High-Dimensional Indexing

A number of techniques have been introduced to address the high-dimensional indexing problem, such as the X-tree [5], and the GC-tree [6]. While these index structures have been shown to increase the range of effective dimensionality, they still suffer performance degradation at higher index dimensionality.

B. Feature Selection

Feature selection techniques [7], [8], [9] are a subset of dimensionality reduction targeted at finding a set of untransformed attributes that best represent the overall data set. These techniques are also focused on maximizing data energy or classification accuracy rather than query response. As a result, selected features may have no overlap with queried attributes.

C. Index Selection

The index selection problem has been identified as a variation of the Knapsack Problem and several papers proposed designs for index recommendations [10], [11], [12], [13], [14], [15] based on optimization rules. These earlier designs could not take advantage of modern database systems' query optimizer. Currently, almost every commercial Relational Database Management System (RDBMS) provides the users with an index recommendation tool based on a query workload and using the query optimizer to obtain cost estimates. A query workload is a set of SQL data manipulation statements. The query workload should be a good representative of the types of queries an application supports.

Microsoft SQL Server's AutoAdmin tool [16], [17], [18] selects a set of indexes for use with a specific data set given a query workload. In the AutoAdmin algorithm, an iterative process is utilized to find an optimal configuration. First, single dimension candidate indexes are chosen. Then a candidate index selection step evaluates the queries in a given query workload and eliminates from consideration those candidate indexes which would provide no useful benefit. Remaining candidate indexes are evaluated in terms of estimated performance improvement and index cost. The process is iterated for increasingly wider multi-column indexes until a maximum index width threshold is reached or an iteration yields no improvement in performance over the last iteration.

Costs are estimated using the query optimizer which is limited to considering those physical designs offered by the DBMS. In the case of SQL Server, single and multi-level B+-trees are evaluated. These index structures can not achieve the same level of result pruning that can be offered by an index technique that indexes multiple dimensions simultaneously (such as R-tree or grid file). As a result the indexes suggested by the tool often do not capture the query performance that could be achieved for multi-dimensional queries.

MAESTRO (METU Automated indEx Selection Tool)[19] was developed on top of Oracle's DBMS to assist the database administrator in designing a complete set of primary and secondary indexes by considering the index maintenance costs based on the valid SQL statements and their usage statistics automatically derived using SQL Trace Facility during a regular database session. The SQL statements are classified by their execution plan and their weights are accumulated. The cost function computed by the query optimizer is used to calculate the benefit of using the index.

For DB2, IBM has developed the DB2Adviser [20] which recommends indexes with a method similar to AutoAdmin with the difference that only one call to the query optimizer is needed since the enumeration algorithm is inside the optimizer itself.

These commercial index selection tools are coupled to physical design options provided by their respective query optimizers and therefore, do not reflect the pruning that could be achieved by indexing multiple dimensions together.

D. Automatic Index Selection

The idea of having a database that can tune itself by automatically creating new indexes as the queries arrive has been proposed [21], [22]. In [21] a cost model is used to identify beneficial indexes and decide when to create or drop an index at runtime. [22] proposes an agent-based database architecture to deal with automatic index creation. Microsoft Research has proposed a physical design alerter [23] to identify when a modification to the physical design could result in improved performance.

III. Approach

A. Problem Statement

In this section we define the problem of index selection for a multi-dimensional space using a query workload.

A query workload W consists of a set of queries that select objects within a specified subspace in the data domain. More formally, we define a workload as follows:

Definition 1: A workload W is a tuple $W = (D, DS, Q)$, where D is the domain, $DS \subseteq D$ is a finite subset (the data set), and Q (the query set) is a set of subsets of DS.
In our case, the domain is $\Re^d$, where d is the dimensionality of the dataset, the instance DS is a set of n tuples $t = \{(a_1, a_2, ..., a_d) | a_i \in \Re\}$, and the query set Q is a set of range queries.

Finding the answers to a query, when no index is present, reduces to scanning all the points in the dataset and testing whether the query conditions are met. In this scenario we can define the *cost* of answering the query as the time it takes to scan the dataset, i.e. the time to retrieve the data pages from disk. The assumption is that the time spent performing I/O dominates the time required to perform the simple bound comparisons. In the case that an index is present, the cost of answering the query can be lower. The index can identify a smaller set potential matching objects and only those data pages containing these objects need to be retrieved from disk. The degree to which an index prunes the potential answer set for a query determines its effectiveness for the query.

Our problem can be defined as finding a set of indexes I, given a multi-dimensional dataset DS, a query workload W, an *optional* indexing constraint C, an *optional* analysis time constraint t_a, that provides the best estimated cost over W. In the context of this problem an index is considered to be the set of attributes that can be used to prune subspace simultaneously with respect to each attribute. Therefore, attribute order has no impact on the amount of pruning possible.

The overall goal of this work is to develop a flexible index selection framework that can be tuned to achieve effective static and online index selection for high-dimensional data under different analysis constraints.

For static index selection, when no constraints are specified, the goal is to recommend the set of indexes that yields the lowest estimated cost for every query in a workload for any query that *can* benefit from an index. In the case when a constraint is specified, either as the minimum number of indexes or a time constraint, we want to recommend a set of indexes within the constraint, from which the queries can benefit the most. When there is a time-constraint, we need to automatically adjust the analysis parameters to increase the speed of analysis.

For online index selection, the goal is to develop a system that can recommend an evolving set of indexes for incoming queries

over time, such that the benefit of index set changes outweighs the cost of making those changes. Therefore, an online index selection system that differentiates between low-cost index set changes and higher cost index set changes and also can make decisions about index set changes based on different cost-benefit thresholds is desirable.

B. Approach Overview

In order to measure the benefit of using a potential index over a set of queries, it is necessary to be able to estimate the cost of executing the queries, with and without the index. Typically, a cost model is embedded into the query optimizer to decide on the query plan, whether the query should be answered using a sequential scan or using an existing index. Instead of using the query optimizer to estimate query cost, we conservatively estimate the number of matches associated with using a given index by using a multi-dimensional histogram abstract representation of the dataset. The histogram captures data correlations between only those attributes that could be represented in a selected index. The cost associated with an index is calculated based on the number of estimated matches derived from the histogram and the dimensionality of the index. Increasing the size of the multi-dimensional histogram enhances the accuracy of the estimate at the cost of abstract representation size.

While maintaining the original query information for later use to determine estimated query cost, we apply one abstraction to the query workload to convert each query into the set of attributes referenced in the query. We perform frequent itemset mining over this abstraction and only consider those sets of attributes that meet a certain support to be potential indexes. By varying the support, we affect the speed of index selection and the ratio of queries that are covered by potential indexes. We further prune the analysis space using association rule mining by eliminating those subsets above a certain confidence threshold. Lowering the confidence threshold improves analysis time by eliminating some lower dimensional indexes from consideration but can result in recommending indexes that cover a strict superset of the queried attributes.

Our technique differs from existing tools in the method we use to determine the potential set of indexes to evaluate and in the quantization-based technique we use to estimate query costs. All of the commercial index wizards work in design time. The Database Administrator (DBA) has to decide when to run this wizard and over which workload. The assumption is that the workload is going to remain static over time and in case it does change, the DBA would collect the new workload and run the wizard again. The flexibility afforded by the abstract representation we use allows it to be used for infrequent, index selection considering a broader analysis space or frequent, online index selection.

In the following two subsections we present our proposed solution for index selection which is used for static index selection and as a building block for the online index selection.

C. Proposed Solution for Index Selection

The goal of the index selection is to minimize the cost of the queries in the workload given certain constraints. Given a query workload, a dataset, the indexing constraints, and several analysis parameters, our framework produces a set of suggested indexes as an output. Figure 1 shows a flow diagram of the index selection framework. Table I provides a list of the notations used in the descriptions.

Symbol	Description
P	Potential Set of Indexes, the set of attribute sets under consideration as a suggested index
Q	Query Set, a representation of the query workload, for each query. It consists of the attribute sets in P that intersect with the query attributes, query ranges, and estimated query costs
H	Multi-Dimensional Histogram, used as a workload-optimized abstract representation of the data set
S	Suggested Indexes, the set of attribute sets currently selected as recommended indexes
i	attribute set currently under analysis
$support$	the minimum ratio of occurrence of an attribute in a query workload to be included in P
$confidence$	the maximum ratio of occurrence of an attribute subset to the occurrence of a set before the subset is pruned from P
$histogram\ size$	the number of bits used to represent a quantized data object

TABLE I
INDEX SELECTION NOTATION LIST

We identify three major components in the index selection framework: the initialization of the abstract representations, the query cost computation, and the index selection loop. In the following subsections we describe these components and the dataflow between them.

1) Initialize Abstract Representations: The initialization step uses a query workload and the dataset to produce a set of Potential Indexes P, a Query Set Q, and a Multi-dimensional Histogram H, according to the support, confidence, and histogram size specified by the user. The description of the outputs and how they are generated is given below.

- Potential Index Set P

The potential index set P is a collection of attribute sets that *could* be beneficial as an index for the queries in the input *query workload*. This set is computed using traditional data mining techniques. Considering the attributes involved in each query from the input *query workload* to be a single transaction, P consists of the sets of attributes that occur together in a query at a ratio greater than the input *support*. Formally, *support* of a set of attributes *A* is defined as:

$$S_A = \frac{\sum_{i=1}^{n} \begin{cases} 1 & \text{if } A \subseteq Q_i \\ 0 & \text{otherwise} \end{cases}}{n}$$

where Q_i is the set of attributes in the i^{th} query and *n* is the number of queries.

For instance, if the input support is 10%, and attributes 1 and 2 are queried together in greater than 10 percent of the queries, then a representation of the set of attributes {1,2} will be included as a potential index. Note that because a subset of an attribute set that meets the support requirement will also necessarily meet the support, all subsets of attribute sets meeting the support will also be included as a potential index (in the example above both the sets {1} and {2} will be included). As the input *support* is decreased, the number of potential indexes increases. Note that our particular system is built independently from a query optimizer, but the sets of attributes appearing in the predicates

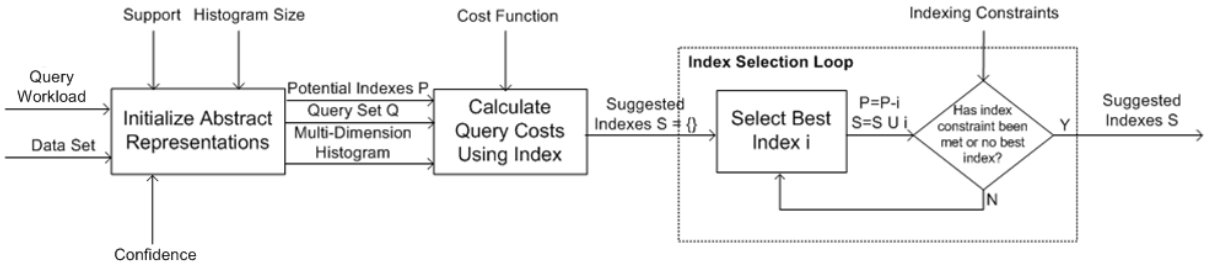

Fig. 1. Index Selection Flowchart

from a query optimizer log could just as easily be substituted for the *query workload* in this step.

If a set occurs nearly as often as one of its subsets, an index built over the subset will likely not provide much benefit over the query workload if an index is built over the attributes in the set. Such an index will only be more effective in pruning data space for those queries that involve only the subset's attributes. In order to enhance analysis speed with limited effect on accuracy, the input *confidence* is used to prune analysis space. Confidence is the ratio of a set's occurrence to the occurrence of a subset.

While data mining the frequent attribute sets in the query workload in determining P, we also maintain the association rules for disjoint subsets and compute the confidence of these association rules. The confidence of an association rule is defined as the ratio that the antecedent (Left Hand Side of the rule) and consequent (Right Hand Side of the rule) appear together in a query given that the antecedent appears in the query. Formally, *confidence* of an association rule {set of attributes A}→{set of attributes B}, where A and B are disjoint, is defined as:

$$C_{A\rightarrow B} = \frac{\sum_{i=1}^{n} \begin{cases} 1 & \text{if } (A \cup B) \subseteq Q_i \\ 0 & \text{otherwise} \end{cases}}{\sum_{i=1}^{n} \begin{cases} 1 & \text{if } A \subseteq Q_i \\ 0 & \text{otherwise} \end{cases}}$$

where Q_i is the set of attributes in the i^{th} query and n is the number of queries.

In our example, if every time attribute 1 appears, attribute 2 also appears then the confidence of {1}→{2} = 1.0. If attribute 2 appears without attribute 1 as many times as it appears with attribute 1, then the confidence {2}→{1} = 0.5. If we have set the *confidence* input to 0.6, then we will prune the attribute set {1} from P, but we will keep attribute set {2}.

We can also set the confidence level based on attribute set cardinality. Since the cost of including extra attributes that are not useful for pruning increases with increased indexed dimensionality, we want to be more conservative with respect to pruning attribute subsets. The *confidence* could take on a value that is dependent on set cardinality.

While the apriori algorithm was appropriate for the relatively low attribute query sets in our domain, a more efficient algorithm such as the FP-Tree [24] could be applied if the attribute sets associated with queries are too large for the apriori technique to be efficient. While it is desirable to avoid examining a high-dimensional index set as a potential index, another possible solution in the case where a large number of attributes are frequent together would be to partition a large closed frequent itemset into disjoint subsets for further examination. Techniques such as CLOSET [25] could be used to arrive at the initial closed frequent itemsets.

- Query Set Q

The query set Q is the abstract representation of the *query workload*. It is initialized by associating the potential indexes that *could* be beneficial for each query with that query. These are the indexes in the potential index set P that share at least one common attribute with the query. At the end of this step, each query has an identified set of possible indexes for that query.

- Multi-Dimensional Histogram H

An abstract representation of the data set is created in order to estimate the query cost associated with using each query's possible indexes to answer that query. This representation is in the form of a multi-dimensional histogram H. A single bucket represents a unique bit representation across all the attributes represented in the histogram. The input *histogram size* dictates the number of bits used to represent each unique bucket in the histogram. These bits are designated to represent only the single attributes that met the input *support* in the input *query workload*. If a single attribute does not meet the *support*, then it can not be part of an attribute set appearing in P. There is no reason to sacrifice data representation resolution for attributes that will not be evaluated. The number of bits that each of the represented attributes gets is proportional to the log of that attribute's support. This gives more resolution to those attributes that occur more frequently in the *query workload*.

Data for an attribute that has been assigned b bits is divided into 2^b buckets. In order to handle data sets with uneven data distribution, we define the ranges of each bucket so that each bucket contains roughly the same number of points. The histogram is built by converting each record in the data set to its representation in bucket numbers. As we process data rows, we only aggregate the count of rows with each unique bucket representation because we are just interested in estimating query cost. Note that the multi-dimensional histogram is based on a scalar quantizer designed on data and access patterns, as opposed to just data in the traditional case. A higher accuracy in representation is achieved by using more bits to quantize the attributes that are more frequently queried.

For illustration, Table II shows a simple multi-dimensional histogram example. This histogram covers 3 attributes and uses 1 bit to quantize attributes 2 and 3, and 2 bits to quantize attribute 1, assuming it is queried more frequently than the other attributes. In this example, for attributes 2 and 3 values from 1-5 quantize to 0, and values from 6-10 quantize to 1. For attribute 1, values 1 and 2 quantize to 00, 3 and 4 quantize to 01, 5-7 quantize to 10, and 8 and 9 quantize to 11. The .'s in the 'value' column denote attribute boundaries (i.e. attribute 1 has 2 bits assigned to it).

Note that we do not maintain any entries in the histogram for bit representations that have no occurrences. So we can not have more histogram entries than records and will not suffer from exponentially increasing the number of potential multi-dimensional histogram buckets for high-dimensional histograms.

Sample Dataset			
A_1	A_2	A_3	Encoding
2	5	5	0000
4	8	3	0110
1	4	3	0000
6	7	1	1010
3	2	2	0100
2	2	6	0001
5	6	5	1010
8	1	4	1100
3	8	7	0111
9	3	8	1101

Histogram	
Value	Count
00.0.0	2
00.0.1	1
01.0.0	1
01.1.0	1
01.1.1	1
10.1.0	2
11.0.0	1
11.0.1	1

TABLE II
HISTOGRAM EXAMPLE

2) Query Cost Calculation: Once generated, the abstract representations of the query set Q and the multi-dimensional histogram H are used to estimate the cost of answering each query using all possible indexes for the query. For a given query-index pair, we aggregate the number of matches we find in the multi-dimensional histogram looking only at the attributes in the query that also occur in the index (bits associated with other attributes are considered to be don't cares in the query matching logic). To estimate the query cost, we then apply a cost function based on the number of matches we obtain using the index and the dimensionality of the index. At the end of this step, our abstract query set representation has estimated costs for each index that could improve the query cost. For each query in the query set representation, we also keep a current cost field, which we initialize to the cost of performing the query using sequential scan. At this point, we also initialize an empty set of suggested indexes S.

- Cost Function

A cost function is used to estimate the cost associated with using a certain index for a query. The cost function can be varied to accurately reflect a cost model for the database system. For example, one could apply a cost function that amortized the cost of loading an index over a certain number of queries or use a function tailored to the type of index that is used. Many cost functions have been proposed over the years. For an R-Tree, which is the index type used for this work, the expected number of data page accesses is estimated in [26] by:

$$A_{nn,mm,FBF} = \left(\sqrt[d]{\frac{1}{C_{eff}}} + 1 \right)^d$$

where d is the dimensionality of the dataset and C_{eff} is the number of data objects per disk page. However, this formula assumes the number of points N approaches to infinity and does not consider the effects of high dimensionality or correlations.

A more recently proposed cost model is given in [27] where the expected number of pages accesses is determined as:

$$A_{r,em,ui}(r) = \left(2r \cdot \sqrt[d]{\frac{N}{C_{eff}}} + 1 - \frac{1}{C_{eff}} \right)^d$$

where r is the radius of the range query, d is the dataset dimensionality, N is the number of data objects, and C_{eff} is the capacity of a data page.

While these published cost estimates can be effective to estimate the number of page accesses associated with using a multi-dimensional index structure under certain conditions, they have certain characteristics that make them less than ideal for the given situation. Each of the cost estimate formulas require a range radius. Therefore the formulas break down when assessing the cost of a query that is an exact match query in one or more of the query dimensions. These cost estimates also assume that data distribution is independent between attributes, and that the data is uniformly distributed throughout the data space.

In order to overcome these limitations, we apply a cost estimate that is based on the actual matches that occur over the multi-dimension histogram over the attributes that form a potential index. The cost model for R-trees we use in this work is given by

$$(d^{(d/2)} * m)$$

where d is the dimensionality of the index and m is the number of matches returned for query matching attributes in the multi-dimensional histogram. Using actual matches eliminates the need for a range radius. It also ties the cost estimate to the actual data characteristics (i.e. incorporates both data correlation between attributes and data distribution, while the published models will produce results that are dependent only on the range radius for a given index structure). The cost estimate provided is conservative in that it will provide a result that is at least as great as the actual number of matches in the database.

By evaluating the number of matches over the set of attributes that match the query, the multi-dimensional subspace pruning that can be achieved using different index possibilities is taken into account. There is additional cost associated with higher dimensionality indexes due to the greater number of overlaps of the hyperspaces within the index structure, and additional cost of traversing the higher dimension structure. A penalty is imposed on a potential index by the dimensionality term. Given equal ability to prune the space, a lower dimensional index will translate into a lower cost.

The cost function could be more complicated in order to more accurately model query costs. It could model query cost with greater accuracy, for example by crediting complete attribute coverage for coverage queries. It could also reflect the appropriate index structures used in the database system, such as B+-trees. We used this particular cost model because the index type was appropriate for our data and query sets, and we assumed that we would retrieve data from disk for all query matches.

3) Index Selection Loop: After initializing the index selection data structures and updating estimated query costs for each potentially useful index for a query, we use a greedy algorithm that takes into account the indexes already selected to iteratively select indexes that would be appropriate for the given query workload and data set. For each index in the potential index set P, we traverse the queries in query set Q that could be improved by that index and accumulate the improvement associated with using that index for that query. The improvement for a given query-index pair is the difference between the cost for using the index and the query's current cost. If the index does not provide any positive benefit for the query, no improvement is accumulated. The potential index i that yields the highest improvement over the

query set Q is considered to be the best index. Index i is removed from potential index set P and is added to suggested index set S. For the queries that benefit from i, the current query cost is replaced by the improved cost.

After each i is selected, a check is made to determine if the index selection loop should continue. The input *indexing constraints* provides one of the loop stop criteria. The indexing constraint could be any constraint such as the number of indexes, total index size, or total number of dimensions indexed. If no potential index yields further improvement, or the *indexing constraints* have been met, then the loop exits. The set of suggested indexes S contains the results of the index selection algorithm.

At the end of a loop iteration, when possible, we prune the complexity of the abstract representations in order to make the analysis more efficient. This includes actions such as eliminating potential indexes that do not provide better cost estimates than the current cost for any query and pruning from consideration those queries whose best index is already a member of the set of suggested indexes. The overall speed of this algorithm is coupled with the number of potential indexes analyzed, so the analysis time can be reduced by increasing the *support* or decreasing the *confidence*.

Different strategies can be used in selecting a best index. The strategy provided assumes a indexing constraint based on the number of indexes and therefore uses the total benefit derived from the index as the measure of index 'goodness'. If the indexing constraint is based on total index size, then benefit per index size unit may be a more appropriate measure. However, this may result in recommending a lower-dimensioned index and later in the algorithm a higher-dimensioned index that always performs better. The recommendation set can be pruned in order to avoid recommending an index that is non-useful in the context of the complete solution.

D. Proposed Solution for Online Index Selection

The online index selection is motivated by the fact that query patterns can change over time. By monitoring the query workload and detecting when there is a change on the query pattern that generated the existing set of indexes, we are able to maintain good performance as query patterns evolve. In our approach, we use control feedback to monitor the performance of the current set of indexes for incoming queries and determine when adjustments should be made to the index set. In a typical control feedback system, the output of a system is monitored and based on some function involving the input and output, the input to the system is readjusted through a control feedback loop. Our situation is analogous but more complex than the typical electrical circuit control feedback system in several ways:

1) Our system input is a set of indexes and a set of incoming queries rather than a simple input, such as an electrical signal.
2) The system output must be some parameter that we can measure and use to make decisions about changing the input. Query performance is the obvious parameter to monitor. However, because lower query performance could be related to other aspects rather than the index set, our decision making control function must necessarily be more complex than a basic control system.
3) We do not have a predictable function to relate system input and output because of the non-determinism associated with new incoming queries. For example, we may have a set of attributes that appears in queries frequently enough that our system indicates that it is beneficial to create an index over those attributes, but there is no guarantee that those attributes will ever be queried again.

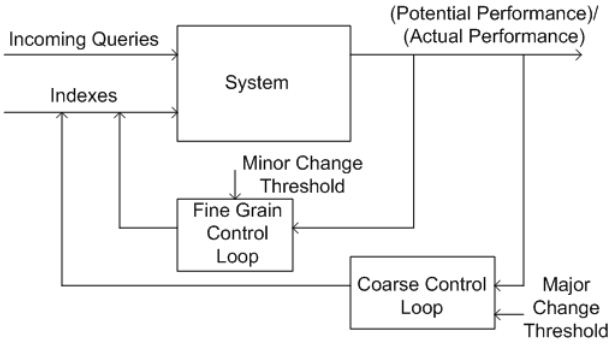

Fig. 2. Dynamic Index Analysis Framework

Symbol	Description
I	Current set of attribute sets used as indexes
I_{new}	Hypothetical set of attribute sets used as indexes
w	window size
W	abstract representation of the last w queries
q	the current query under analysis
P	Current Potential Indexes, the set of attribute sets in consideration to be indexes before q arrives
P_{new}	New Potential Indexes, the set of attribute sets in consideration to be indexes after q arrives
i_q	the attribute set estimated to be the best index for query q

TABLE III
NOTATION LIST FOR ONLINE INDEX SELECTION

Control feedback systems can fail to be effective with respect to response time. The control system can be too slow to respond to changes, or it can respond too quickly. If the system is too slow, then it fails to cause the output to change based on input changes in a timely manner. If it responds too quickly, then the output overshoots the target and oscillates around the desired output before reaching it. Both situations are undesirable and should be designed out of the system.

Figure 2 represents our implementation of dynamic index selection. Our system input is a *set of indexes* and a *set of incoming queries*. Our system simulates and estimates costs for the execution of incoming queries. System output is the ratio of potential system performance to actual system performance in terms of database page accesses to answer the most recent queries. We implement two control feedback loops. One is for fine grain control and is used to recommend minor, inexpensive changes to the index set. The other loop is for coarse control and is used to avoid very poor system performance by recommending major index set changes. Each control feedback loop has decision logic associated with it.

1) System Input: The system input is made up of new incoming queries and the current set of indexes I, which is initialized to be the suggested indexes S from the output of the initial index selection algorithm. For clarity, a notation list for the online index selection is included as Table III.

2) System: The system simulates query execution over a number of incoming queries. The abstract representation of the last w queries stored as W , where w is an adjustable window

size parameter. W is used to estimate performance of a hypothetical set of indexes I_{new} against the current index set I. This representation is similar to the one kept for query set Q in the static index selection. In this case, when a new query q arrives, we determine which of the current indexes in I most efficiently answers this query and replace the oldest query in W with the abstract representation of q. We also incrementally compute the attribute sets that meet the input *support* and *confidence* over the last w queries. This information is used in the control feedback loop decision logic. The system also keeps track of the current potential indexes P, and the current multi-dimensional histogram H.

3) System Output: In order to monitor the performance of the system, we compare the query performance using the current set of indexes I to the performance using a hypothetical set of indexes I_{new}. The query performance using I is the summation of the costs of queries using the best index from I for the given query. Consider the possible new indexes P_{new} to be the set of attribute sets that currently meet the input *support* and *confidence* over the last w queries. The hypothetical cost is calculated differently based on the comparison of P and P_{new}, and the identified best index i_q from P or P_{new} for the new incoming query:

1) $P = P_{new}$ and i is in I. In this case we bypass the control loops since we could do no better for the system by changing possible indexes.
2) $P = P_{new}$ and i is not in I. We recompute a new set of suggested indexes I_{new} over the last w queries. The hypothetical cost is the cost over the last w queries using I_{new}.
3) $P \neq P_{new}$ and i is in I. In this case we bypass the control loops since we could do no better for the system by changing possible indexes.
4) $P \neq P_{new}$ and i is not in I. We traverse the last w queries and determine those queries that could benefit from using a new index from P_{new}. We compute the hypothetical cost of these queries to be the real number of matches from the database. Hypothetical cost for other queries is the same as the real cost.

The ratio of the hypothetical cost, which indicates potential performance, to the actual performance is used in the control loop decision logic.

4) Fine Grain Control Loop: The fine grain control loop is used to recommend low cost, minor changes to index set. This loop is entered in case 2 as described above when the ratio of hypothetical performance to actual performance is below some input *minor change threshold*. Then the indexes are changed to I_{new}, and appropriate changes are made to update the system data structures. Increasing the input *minor change threshold* causes the frequency of minor changes to also increase.

5) Coarse Control Loop: The coarse control loop is used to recommend more costly, but changes with greater impact on future performance to the index set. This loop is entered in case 4 as described above when the ratio of hypothetical performance to actual performance is below some input *major change threshold*. Then the static index selection is performed over the last w queries, abstract representations are recomputed, and a new set of suggested indexes I_{new} is generated. Appropriate changes are made to update the system data structures to the new situation. Increasing the input *major change threshold* increases the frequency of major changes.

E. System Enhancements

In the following subsections, we present two system enhancements that provide further robustness and scalability to the framework.

1) Self Stabilization: Control feedback systems can be either too slow or too responsive in reacting to input changes. In our application, a system that is slow to respond results in recommending useful indexes long after they first could have a positive effect. It could also fail to recommend potentially useful indexes if thresholds are set so that the system is insensitive to change. A system that is too responsive can result in system instability, where the system continuously adds and drops indexes.

The system performance and rate of index change can be monitored and used in order to tune the control feedback system itself. If the actual number of query results is much lower than the estimated cost using the recommended index set over a window of queries, this indicates a non-responsive system. When this condition is detected, the system can be made more responsive by cutting the window size. This increases the probability that P_{new} will be different from P, and we can reperform to analysis applying a reduced *support*. This gives us a more complete P with respect to answering a greater portion of queries.

If the frequency of index change is too high with little or no improvement in query performance, an oversensitive system or unstable query pattern is indicated. We can reduce the sensitivity by increasing the window size or increasing the support level during recomputation of new recommended indexes.

2) Partitioning the Algorithm: The system can also be applied to environments where the database is partitioned horizontally or vertically at different locations. A solution at one extreme is to maintain a centralized index recommendation system. This would maintain the abstract representation and collect global query patterns over the entire system. Recommended indexes would be determined based on global trends. This approach would allow for the creation of indexes that maximize pruning over the global set of dimensions. However, it would not optimize query performance at the site level. A single set of indexes would be recommended for the entire system.

At the other extreme would be to perform the index recommendation at each site. In this case, a different abstract representation would be built based on the data at each specific site given requests to the data at that site. Indexes would be recommended based on the query traffic to the data at that site. This allows tight control of the indexes to reflect the data at each site. However, index-based pruning based on attributes and data at multiple locations would not be possible. This approach also requires traffic to each site that contains potentially matching data.

A hybrid approach can provide the benefits of index recommendation based on global data and query patterns while optimizing the index set at different requesting locations. A global set of indexes can be centrally recommended based on a global abstract representation of the data set with low support thresholds (the centralized location will store a larger set of globally good indexes). The query patterns emanating from each site can be mined in order to find which of those global indexes would be appropriate to maintain at the local site, eliminating some traffic.

IV. Empirical Analysis

A. Experimental Setup

Data Sets

Several data sets were used during the performance of experiments. The variation in data sets is intended to show the applicability of our algorithm to a wide range of data sets and to measure the effect that data correlation has on results. Data sets used include:

- *random* - a set of 100,000 records consisting of 100 dimensions of uniformly distributed integers between 0 and 999. The data is not correlated.
- *stocks* - a set of 6500 records consisting of 360 dimensions of daily stock market prices. This data is extremely correlated.
- *mlb* - a set of 33619 records of major league pitching statistics from between the years of 1900 and 2004 consisting of 29 dimensions of data. Some dimensions are correlated with each other, while others are not at all correlated.

Analysis Parameters

The effect of varying several analysis input parameters including support, multi-dimensional histogram size, and online indexing control feedback decision thresholds was analyzed. Unless otherwise specified, the $confidence$ parameter for the experiments is 1.0.

Query Workloads

It is desirable to explore the general behavior of database interactions without inadvertently capturing the coupling associated with using a specific query history on the same database. Therefore, query workload files were generated by merging synthetic query histories and query histories from real-world applications with different data sets. Histories and data were merged by taking a random record from a data set and the numerical identifier of the attributes involved in the synthetic or historical query in order to generate a point query. So, if a historical query involved the 3rd and 5th attribute, and the n^{th} record was randomly selected from the data set, a SELECT type query is generated from the data set where the 3rd attribute is equal to the value of the 3rd attribute of the n^{th} record and the 5th attribute is equal to the value of 5th attribute of the n^{th} record. This gives a query workload that reflects the attribute correlations within queries, and has a variable query selectivity. Unless otherwise stated each query in experiments is a point query with respect to the attributes covered by the query. Attributes that are not covered by the query can be any value and still match the query.

These query histories form the basis for generating the query workloads used in our experiments:

1) synthetic - 500 randomly generated queries. The distribution of the queries over the first 200 queries is 20% involve attributes {1,2,3,4} together, 20% {5,6,7}, 20%, {8,9}, and the remaining queries involve between 1 and 5 attributes that could be any attribute. Over the last 300 queries, the distribution shifts to 20% covering attributes {11,12,13,14}, 20% {15,16,17}, 20% {18,19}, and the remaining 40% are between 1 to 5 attributes that could be any attribute.
2) clinical - 659 queries executed from a clinical application. The query distribution file has 64 distinct attributes.
3) hr - 35,860 queries executed from a human resources application. The query distribution file has 54 attributes. Due to the size of this query set, some initial portion of the queries are used for some experiments.

B. Experimental Results

1) Index Selection with Relaxed Constraints: Figure 3 shows how accurately the proposed static index selection technique can perform with relaxed analysis constraints. For this scenario, a low support value, 0.01, is used. There are no constraints placed on the number of selected indexes and 256 bits are used for the multi-dimension histogram. Figure 3 shows the cost of performing a sequential scan over 100 queries using the indicated data sets, the estimated cost of using recommended indexes, and the true number of answers for the set of queries. For comparative purposes, costs for indexes proposed by AutoAdmin and a naive algorithm are also provided. The naive algorithm uses indexes for the first n most frequent itemsets in the query patterns, where n is the number of indexes suggested by the proposed algorithm. Note that the cost for sequential scan is discounted to account for the fact that sequential access is significantly cheaper than random access. A factor of 10 is applied as the ratio of random access cost to sequential access cost. While the actual ratio of a given environment is variable, regardless of any reasonable factor used, the graph will show the cost of using our indexes much closer to the ideal number of page accesses than the cost of sequential scan.

Table IV compares the proposed index selection algorithm with relaxed constraints against SQL Server's index selection tool, AutoAdmin [16], using the data sets and query workloads indicated.

Data Set/ Workload	Tool	Analysis Time(s)	% Queries Improved	Number of Indexes
stock/ clinical	AutoAdmin	450	100	23
	Proposed	110	100	18
stock/ hr	AutoAdmin	338	100	20
	Proposed	160	100	16
mlb/ clinical	AutoAdmin	15	0	0
	Proposed	522	87	16

TABLE IV

COMPARISON OF PROPOSED INDEX SELECTION ALGORITHM WITH AUTOADMIN IN TERMS OF ANALYSIS TIME AND % QUERIES IMPROVED

For the *stock* dataset using both the *clinical* and *hr* workloads, both algorithms suggest indexes which will improve all of the queries. Since the selectivity of these queries is low (the queries return a low number of matches using any index that contains a queried attribute), the amount of the query improvement will be very similar using either recommended index set. The proposed algorithm executes in less time and generates indexes which are more representative of the query patterns in the workload and allow for greater subspace pruning. For example, the most common query in the *clinical* workload is to query over both attributes 11 and 44 together. SQL Server's index recommendations are all single-dimension indexes for all attributes that appear in the workload. However, our first index recommendation is a 2-dimension index built over attributes 11 and 44.

For the *mlb* dataset, SQL Server quickly recommended no indexes. Our index selection takes longer in this instance, but finds indexes that improve 87 % of the queries. These are all the queries that have selectivities low enough that an index can be beneficial. It should be noted that the indexes selected by AutoAdmin are appropriate based on the cost models for the underlying index structure used in SQL Server. Since the underlying index type for SQL Server is B+-trees, which do not index multiple dimensions

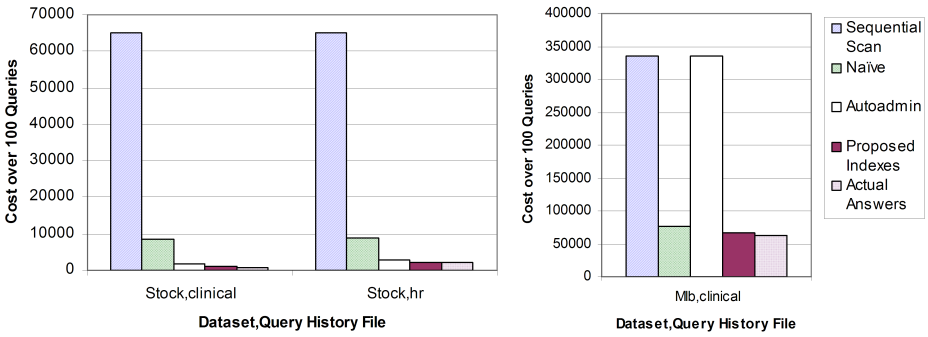

Fig. 3. Costs in Data Object Accesses for Ideal, Sequential Scan, AutoAdmin, Naive, and the Proposed Index Selection Technique using Relaxed Constraints

Support	Query/Index Pairs			Total Improvement		
Confidence	100	50	0	100	50	0
2	229	229	90	57710	57710	57603
4	198	198	85	55018	55018	55001
6	185	185	73	46924	46924	46907
8	178	178	66	42533	42533	42516
10	170	170	58	37527	37527	37510

TABLE V

COMPARISON OF ANALYSIS COMPLEXITY AND QUERY PERFORMANCE AS $support$ AND $confidence$ VARY, STOCK DATASET, CLINICAL WORKLOAD

concurrently, the single dimension indexes that are recommended are the least costly possible indexes to use for the query set. For this particular experiment, a single dimensional index is not able to prune the subspace enough to make the application of that index worthwhile compared to sequential scan.

2) Effect of Support and Confidence: Table V presents results on analysis complexity and expected query improvement as $support$ and $confidence$ are varied. The results are shown for the stock data set over the clinical query workload. Results show the total number of query-index pairs analyzed over the query set in the index selection loop and the total estimated improvement in query performance in terms of data objects accessed over the query set as the index selection parameters vary. As $confidence$ decreases, we maintain fewer potential indexes in P and need to analyze fewer attribute sets per index. This decreases analysis time but shows very little effect on overall performance.

Using a confidence level of 0% is equivalent to using the maximal itemsets of attributes that meet support criteria as recommended indexes. For this example, the strategy yields nearly identical estimated cost although only 34-44% of the query/index pairs need to be evaluated.

3) Baseline Online Indexing Results: A number of experiments were performed to demonstrate the effectiveness and characteristics of the adaptive system. In each of the experiments that show the performance of the adaptive system over a sequence of queries, an initial set of indexes is recommended based on the first 100 queries given the stated analysis parameters. New queries are evaluated, and depending on the parameters of the adaptive system, changes are made to index set. These index set changes are considered to take place before the next query. The estimated cost of the last 100 queries given the indexes available at the time of the query are accumulated and presented. This demonstrates the evolving suitability of the current index set to incoming queries.

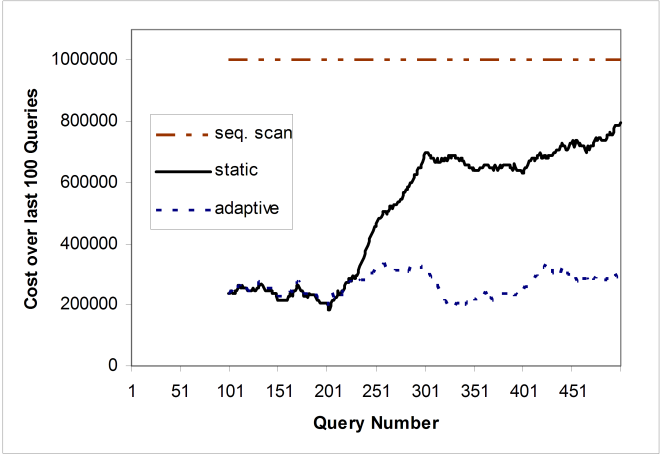

Fig. 4. Baseline Comparative Cost of Adaptive versus Static Indexing, *random* Dataset, *synthetic* Query Workload

Figures 4 through 6 show a baseline comparison between using the query pattern change detection and modifying the indexes and making no change to the initial index set. A number of data set and query history combinations are examined. The baseline parameters used are 5% support, 128 bits for each multi-dimension histogram entry, a window size of 100, an indexing constraint of 10 indexes, a major change threshold of 0.9 and a minor change threshold of 0.95.

Figure 4 shows the comparison for the *random* data set using the *synthetic* query workload. At query 200, this workload drastically changes, and this is evident in the query cost for the static system. The performance gets much worse and does not get better for the static system. For the online system, the performance degrades a little when the query patterns are changing and then improve again once the indexes have changed to match the new query patterns.

The *synthetic* query workload was generated specifically to show the effect of a changing query pattern. Figure 5 shows a comparison of performance for the *random* data set using the *clinical* query workload. This real query workload also changes substantially before reverting back to query patterns similar to those on which the static index selection was performed. Performance for the static system degrades substantially when the patterns change until the point that they change back. The adaptive

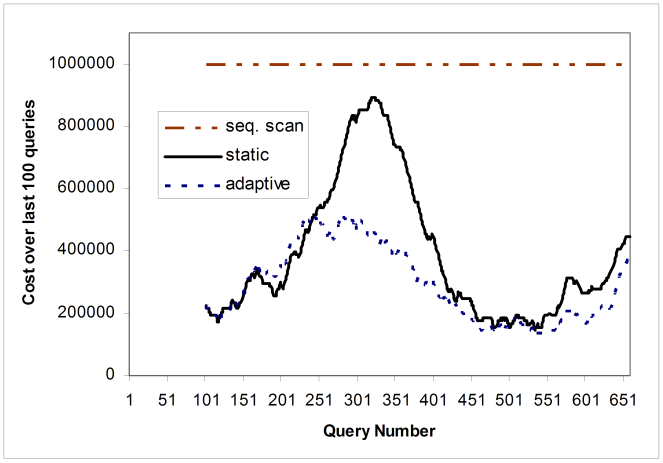

Fig. 5. Baseline Comparative Cost of Adaptive versus Static Indexing, *random* Dataset, *clinical* Query Workload

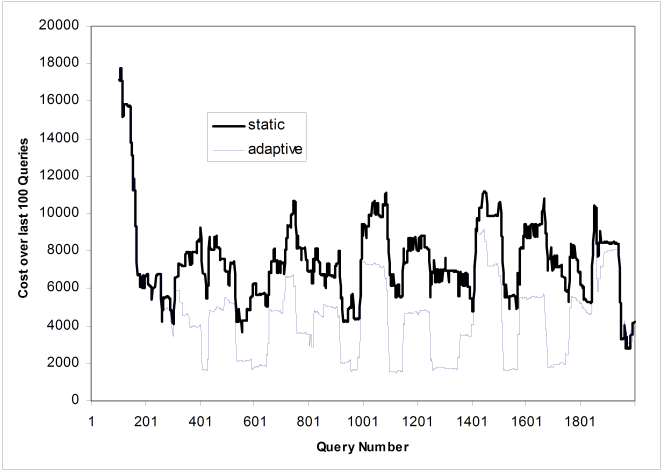

Fig. 6. Baseline Comparative Cost of Adaptive versus Static Indexing, *stock* Dataset, *hr* Query Workload

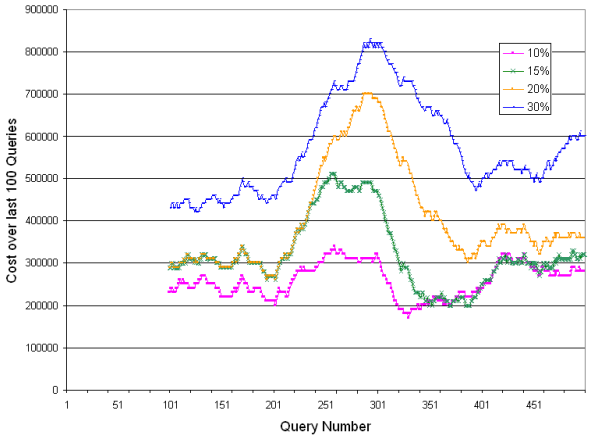

Fig. 7. Comparative Cost of Online Indexing as Support Changes, *random* Dataset, *synthetic* Query Workload

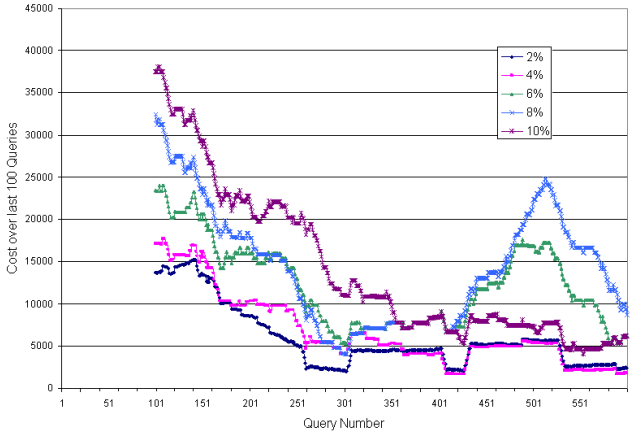

Fig. 8. Comparative Cost of Online Indexing as Support Changes, *stock* Dataset, *hr* Query Workload

system is better able to absorb the change.

Figure 6 shows the comparison for the *stock* data set using the first 2000 queries of the *hr* query workload. The adaptive system shows consistently lower cost than the static system and in places significantly lower cost for this real query workload.

The effect of range queries was also explored. The *random* data set was examined using the *synthetic* workload using ranges instead of points. As the ranges for a given attribute increased from a point value to 30% selectivity, the cost saved for top 3 proposed indexes (which were the index sets [1,2,3,4], [5,6,7], and [8,9] for all ranges), the overall cost saved decreased. This is due to the increase in the size of the answer set. When the range for each attribute reached 30%, no index was recommended because the result set became large enough that sequential scan was a more effective solution.

4) Online Index Selection versus Parametric Changes: Changing online index selection parameters changes the adaptive index selection in the following ways:

Support - decreasing the support level has the effect of increasing the potential number of times the index set will be recalculated. It also makes the recalculation of the recommended indexes themselves more costly and less appropriate for an online setting. Figure 7 shows the effect of changing support on the online indexing results for the *random* data set and *synthetic* query workload, while Figure 8 shows the effect on the *stock* data set using the first 600 queries of *hr* query workload. These graphs were generated using the baseline parameters and varying support levels. As expected, lower support levels translate to lower initial costs because more of the initial query set are covered by some beneficial index. For the *synthetic* query workload, when the patterns changed, but do not change again (e.g. queries 100-200 in Figure 7), lower support levels translated to better performance. However, for the *hr* query workload, the 10% support threshold yields better performance than the 6% and 8% thresholds for some query windows. The frequent itemsets change more frequently for lower support levels, and these changes dictate when decisions occur. These runs made decisions at points that turned out to be poor decisions, such as eliminating an index that ended up being valuable. Little improvement in cost performance is achieved between 4% support and 2% support. Over-sensitive control feedback can degrade actual performance, independent of the extra overhead that oversensitive control causes.

Online Indexing Control Feedback Decision Thresholds - increasing the thresholds decreases the response

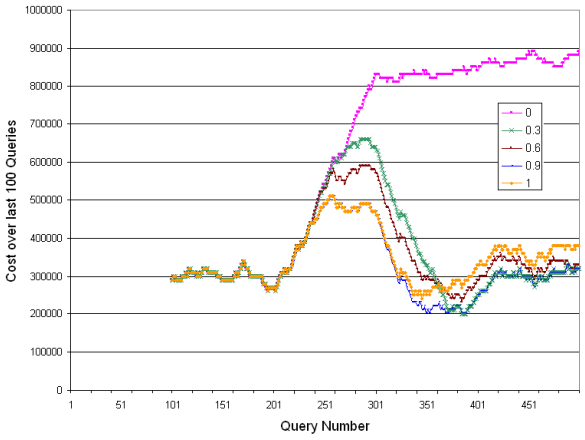

Fig. 9. Comparative Cost of Online Indexing as Major Change Threshold Changes, *random* Dataset, *synthetic* Query Workload

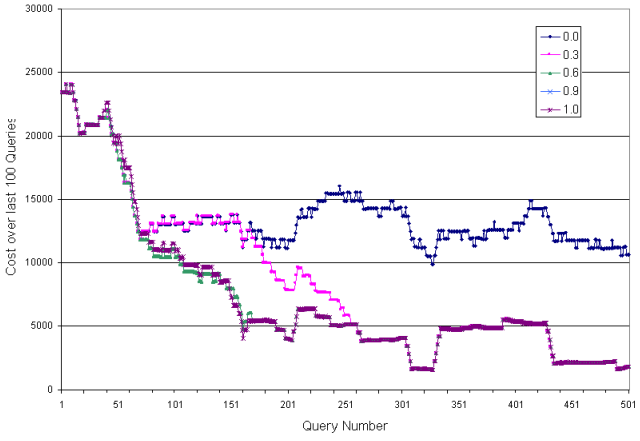

Fig. 10. Comparative Cost of Online Indexing as Major Change Threshold Changes, *stock* Dataset, *hr* Query Workload

time of affecting change when query pattern change does occur. It also has the effect of increasing the number of times the costly reanalysis occurs. In a real system, one would need to balance the cost of continued poor performance against the cost of making an index set change. Figure 9 shows the effect of varying the major change threshold in the coarse control loop for the *random* data set and *synthetic* query workload. Figure 10 shows the effect of changing the major change threshold for the *stock* data set and *hr* query workload. These graphs were generated using the baseline parameters (except that the *random* graph uses a support of 10%), and only varying the major change threshold in the coarse control loop. The major change threshold is varied between 0 and 1. Here, a value of 0 translates to never making a change, and a value of 1 means making a change whenever improvement is possible. The graphs show no real benefit once the major change threshold increases (and therefore frequency of major changes) beyond 0.6. Figure 9 shows that a value of 1 or 0.9 are best in terms of response time when a change occurs, but a value of 0.3 shows the best performance once the query patterns have stabilized. This indicates that this value should be carefully tuned based on the expected frequency of query pattern changes.

Fig. 11. Comparative Cost of Online Indexing as Multi-Dimensional Histogram Size Changes, *random* Dataset, *synthetic* Query Workload

Multi – dimensional Histogram Size - increasing the number of bits used to represent a bucket in the multi-dimensional histogram improves analysis accuracy at the cost of histogram generation time and space. Experiments demonstrated that if the representation quality of the histogram was sufficient, very little benefit was achieved through greater resolution. Figure 11 shows an example of the effect of varying the multi-dimensional histogram size. Using a 32 bit size for each unique histogram bucket yields higher costs than by using greater histogram resolution. One reason for this is that artificially higher costs are calculated because of the lower histogram resolution. As well, some beneficial index changes are not recommended because of these overly conservative cost estimates.

V. Conclusions

A flexible technique for index selection is introduced that can be tuned to achieve different levels of constraints and analysis complexity. A low constraint, more complex analysis can lead to more accurate index selection over stable query patterns. A more constrained, less complex analysis is more appropriate to adapt index selection to account for evolving query patterns. The technique uses a generated multi-dimension histogram to estimate cost, and as a result is not coupled to the idiosyncrasies of a query optimizer, which may not be able to take advantage of knowledge about correlations between attributes. Indexes are recommended in order to take advantage of multi-dimensional subspace pruning when it is beneficial to do so.

These experiments have shown great opportunity for improved performance using adaptive indexing over real query patterns. A control feedback technique is introduced for measuring performance and indicating when the database system could benefit from an index change. By changing the threshold parameters in the control feedback loop, the system can be tuned to favor analysis time or pattern change recognition. The foundation provided here will be used to explore this tradeoff and to develop an improved utility for real-world applications. The proposed technique affords the opportunity to adjust indexes to new query patterns. A limitation of the proposed approach is that if index set changes are not responsive enough to query pattern changes then the control feedback may not affect positive system changes. However, this can be addressed by adjusting control sensitivity or

by changing control sensitivity over time as more knowledge is gathered about the query patterns.

From initial experimental results, it seems that the best application for this approach is to apply the more time consuming no-constraint analysis in order to determine an initial index set and then apply a lightweight and low control sensitivity analysis for the online query pattern change detection in order to avoid or make the user aware of situations where the index set is not at all effective for the new incoming queries.

In this paper, the frequency of index set change has been affected through the use of the online indexing control feedback thresholds. Alternatively, the frequency of performance monitoring could be adjusted to achieve similar results and to appropriately tune the sensitivity of the system. This monitoring frequency could be in terms of either a number of incoming queries or elapsed time.

The proposed online change detection system utilized two control feedback loops in order to differentiate between inexpensive and more time consuming system changes. In practice, the fine grain control threshold was not triggered unless we contrived a situation such that it would be triggered. The kinds of low cost changes that this threshold would trigger are not the kinds of changes that make enough impact to be that much better than the existing index set. This would change if the indexing constraints were very low, and one potential index is now more valuable than a suggested index.

Index creation is quite time-consuming. It is not feasible to perform real-time analysis of incoming queries and generate new indexes when the patterns change. Potential indexes could be generated prior to receiving new queries, and when indicated by online analysis, moved to *active* status. This could mean moving an index from local storage to main memory, or from remote storage to local storage depending on the size of the index. Additionally, the analysis could prompt a server to create a potential index as the analysis becomes aware that such an index is useful, and once it is created, it could be called on by the local machine.

ACKNOWLEDGMENT

We would like to thank the anonymous TKDE reviewers for their many insightful and helpful comments. This research is partially supported by US National Science Foundation grants III-054713, CNS-0403342, and OCI-0619041.

REFERENCES

[1] R. Bellman, *Adaptive Control Processes: A Guided Tour*. Princeton University Press, 1961.

[2] R. Weber, H.-J. Schek, and S. Blott, "A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces," in *Proceedings of the 24th International Conference on Very Large Databases*, 1998, pp. 194–205.

[3] S. Ponce, P. M. Vila, and R. Hersch, "Indexing and selection of data items in huge data sets by constructing and accessing tag collections," in *Proceedings of the 19th IEEE Symposium on Mass Storage Systems and Tenth Goddard Conf. on Mass Storage Systems and Technologies*, 2002.

[4] A. Shoshani, L. Bernardo, H. Nordberg, D. Rotem, and A. Sim, "Multi-dimensional indexing and query coordination for tertiary storage management," in *Proceedings of the 11th International Conference on Scientific and Statistical Data(SSDBM)*, 1999.

[5] S. Berchtold, D. Keim, and H. Kriegel, "The x-tree: An index structure for high-dimensional data," in *Proceedings of the International Conference on Very Large Data Bases*, 1996, pp. 28–39.

[6] C.-W. C. Guang-Ho Cha, "The gc-tree: a high-dimensional index structure for similarity search in image databases," *IEEE Transactions on MultiMedia*, vol. 4, no. 2, pp. 235–247, Jun. 2002.

[7] A. Blum and P. Langley, "Selection of relevant features and examples in machine learning," *AI*, 1997.

[8] R. Kohavi and G. John, "Wrappers for feature subset selection," *AI*, 1997.

[9] I. Guyon and A. Elissef, "An introduction to variable and feature selection," *Journal of Machine Learning Research*, 2003.

[10] M. Ip, L. Saxton, and V. Raghavan, "On the selection of an optimal set of indexes." *IEEE Transactions on Software Engineering*, 1983.

[11] K. Whang, "Index selection in relational databases," in *International Conference on Foundations on Data Organization (FODO)*, Kyoto, Japan, 1985.

[12] E. Barucci, R. Pinzani, and R. Sprugnoli, "Optimal selection of secondary indexes." *IEEE Transactions on Software Engineering*, 1990.

[13] M. Frank, E. Omiecinski, and S. Navathe, "Adaptive and automated index selection in rdbms," in *International Conference on Extending Database Technologhy (EDBT)*, Vienna, Austria, 1992.

[14] S. Choenni, H. Blanken, and T. Chang, "On the selection of secondary indexes in relational databases." *Data and Knowledge Engineering*, 1993.

[15] A. Capara, M. Fischetti, and D. Maio, "Exact and approximate algorithms for the index selection problem in physical database design." *IEEE Transactions on Knowledge and Data Engineering*, 1995.

[16] S. Chaudhuri and V. Narasayya, "Autoadmin 'what-if' index analysis utility," in *Proceedings ACM SIG-MOD Conference*, 1998, pp. 367–378.

[17] S. Chaudhuri and V. R. Narasayya, "An efficient cost-driven index selection tool for microsoft SQL server," in *The VLDB Journal*, 1997, pp. 146–155. [Online]. Available: citeseer.ist.psu.edu/chaudhuri97efficient.html

[18] S. Agrawal, S. Chaudhuri, L. Kollár, A. P. Marathe, V. R. Narasayya, and M. Syamala, "Database tuning advisor for microsoft sql server 2005." in *VLDB*, 2004, pp. 1110–1121.

[19] A. Dogac, A. Y. Erisik, and A. Ikinci, "An automated index selection tool for oracle7: Maestro 7." TUBITAK Software Research and Development Center, Technical Report LBNL/PUB-3161, 1994.

[20] G. Valentin, M. Zuliani, D. Zilio, G. Lohman, and A. Skelley, "Db2 advisor: An optimizer smart enough to recommend its own indexes," in *Proceedings International Conference Data Engineering*, 2000.

[21] S. Kai-Uwe, E. Schallehn, and I. Geist, "Autonomous query-driven index tuning," in *International Database Engineering & Applications Symposium*, Coimbra, Portugal, 2004.

[22] R. L. D. C. Costa and S. Lifschitz, "Index self-tuning with agent-based databases," in *XXVIII Latin-American Conference on Informatics (CLIE)*, Montevideo, Uruguay, 2002.

[23] N. Bruno and S. Chaudhuri, "To tune or not to tune? a lightweight physical design alerter." in *VLDB*, 2006, pp. 499–510.

[24] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," in *2000 ACM SIGMOD Intl. Conference on Management of Data*, W. Chen, J. Naughton, and P. A. Bernstein, Eds. ACM Press, 05 2000, pp. 1–12. [Online]. Available: citeseer.ist.psu.edu/han99mining.html

[25] J. Pei, J. Han, and R. Mao, "CLOSET: An efficient algorithm for mining frequent closed itemsets," in *ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, 2000, pp. 21–30. [Online]. Available: citeseer.ist.psu.edu/pei00closet.html

[26] C. Faloutsos, T. Sellis, and N. Roussopoulos, "Analysis of object oriented spatial access methods," in *SIGMOD '87: Proceedings of the 1987 ACM SIGMOD international conference on Management of data*. New York, NY, USA: ACM Press, 1987, pp. 426–439.

[27] C. Bohm, "A cost model for query processing in high dimensional data spaces," *ACM Trans. Database Syst.*, vol. 25, no. 2, pp. 129–178, 2000.

Michael Gibas is a PhD candidate in the Department of Computer Science and Engineering at The Ohio State University. He currently works in the Database Research Group and his research interest is efficient access to large and high-dimensional data sets. His projects include indexing data based on access patterns, developing techniques to efficiently handle optimization queries, and providing database support for scientific research projects.

Guadalupe Canahuate is a PhD Candidate at The Ohio State University. She received her MS degree in 2003 in Computer Science and Engineering from The Ohio State University and currently works in the Database Research Group supporting scientific applications and enhancing bitmap indexes. Her research interests are in the area of high dimensional data management and indexing.

Hakan Ferhatosmanoglu is an associate professor of computer science and engineering at The Ohio State University. He received the PhD degree in 2001 from the Computer Science Department at the University of California, Santa Barbara, and worked as an intern at AT & T Research Labs. His research interest is developing data management systems for multimedia, scientific, and biomedical applications. He leads projects on microarray and clinical trial databases, online compression and analysis of multiple data streams, and high-performance databases for multidimensional data repositories. Dr. Ferhatosmanoglu is a recipient of the Early Career Principal Investigator award from the US Department of Energy and the Early Career award (CAREER) from the US National Science Foundation (NSF).