Monitoring of Changing Query Workloads for Online Index Recommendations

Michael Gibas, Guadalupe Canahuate, Hakan Ferhatosmanoglu

The authors are with The Ohio State University

Abstract

Because of the oft-cited 'curse of dimensionality', high-dimensional indexes do not work to efficiently access very large multi-dimensional databases. A potential solution is to utilize index selection tools to generate multiple lower dimensional indexes. However, tools developed to perform this function are time-consuming to run and are therefore run statically at times determined by a database administrator. As such, these tools are not responsive to changes in query workload patterns, and if the sets of attributes queried evolve so that they differ from the sets of attributes in the index set, query performance will significantly degrade. To address these issues, we introduce a parameterizable technique to statically and 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 either the introduction of a new index, the replacement of an existing index, or the construction of an entirely new index set. 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. We also track query performance over query sets that change distribution over time, to measure the effectiveness of the query pattern change detection and dynamic index recommendation algorithm.

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. High dimensional indexing and dimensionality reduction have been two popular approaches to handle query processing in high dimensional databases. Even the most effective approaches of these categories suffer from inherent problems. Performance of index structures is subject to Bellman's curse of dimensionality [4] and degrades rapidly as the number of dimensions increases. 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. Novel directions are needed to effectively access data from a high dimensional database without suffering from the inherent dimensionality problems.

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 generate data with up to 500 attributes at the rate of 20 to 40 per second [34]. However, the search criterion typically consists of 10 to 30 parameters. Another example is High Energy Physics (HEP) experiments [39] 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 with the novelty that the reduction is specifically designed for reducing query response times, rather than maintaining data energy as is the case in traditional approaches. Our reduction considers 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.

Several researchers have addressed the indexing problem by developing tools that recommend a set of indexes based on a query workload. However, query access patterns may change over time becoming completely disjoint from the patterns on which the low dimensional indexes were 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.

Static 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 dynamic 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 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. To the best of our knowledge, this is the first paper that incorporates control feedback to affect dynamic index selection under query workload changes.
3) Presentation of a novel data quantization technique optimized for query workloads.

4) Presentation of a partial solution to the ‘curse of dimensionality’ using a dimensionality reduction technique based on query workloads.
5) 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 framework. Section IV presents the empirical analysis. We conclude in Section V.

II. Related Work

A. Dimensionality Reduction

A common approach for dimensionality reduction is to use linear-algebraic methods, such as Karhunen-Loeve Transformation (KLT) or SVD [25], [28], [21], [38], or applications of mathematical transforms such as the Discrete Fourier (DFT), Discrete Cosine (DCT), or Wavelet Transforms [32], [23], [7], and to keep a small subset of transformed dimensions [1]. These transforms and other similar transforms are used in several application areas including signal and image processing [20], [40], [37], [46], [31], [15], time-series databases [35], [36], [47], [26], approximate query processing [24], [8], [44], [18], selectivity estimation [29], [27], and data streams [30], [33]. All these techniques are based on the distribution of the data and do not take advantage of the query patterns.

B. Index Selection

The index selection problem has been identified as a variation of the Knapsack Problem and several papers proposed designs for index recommendations [19], [45], [3], [16], [12], [6] 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 [9], [10], [2] 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. MAESTRO (METU Automated indEx Selection Tool)[14] 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 [42] 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.

C. Automatic Index Selection

The idea of having a database that can tune itself by automatically creating new indexes as the queries come has been proposed [22], [13]. In [22] a cost model is used to identify beneficial indexes and decide when to create or drop an index at runtime. [13] proposes an agent-based database architecture to deal with automatic index creation.

D. Dynamic Frequent Itemset Mining

Many algorithms have been developed recently to addressed the problem of incrementally mining frequent itemsets [41], [43], [11], [17]. The abstract representation of the query set we use allows us to apply incremental frequent itemset mining to perform query pattern change detection. We are using the ZIGZAG approach [43] with update rate of 1.

III. Approach

A. Approach Goals

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 the application of static index selection, when our tool has no constraints applied, we want to recommend the best possible index for each query in a workload for any query that *can* benefit from an index. When we constrain the number of indexes that our system recommends, we want to recommend a set of indexes within the constraint, such that the overall query set in the workload achieves the greatest benefit. In order to be able to operate in time-constrained situations, we would like the ability to adjust analysis parameters to increase the speed of analysis.

In terms of online index selection, we are striving 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 the index set changes. Therefore, we want a dynamic 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.

B. Approach Overview

Like existing tools, our technique uses a cost-based function to evaluate the benefits of using a potential index on a set of queries. Our technique differs from other 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. These differences allow the analysis to be tuned to affect the speed and accuracy of the analysis.

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 level to be potential indexes. By varying this support level, we affect the speed of index selection and the ratio of queries that are covered by potential indexes.

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 cost associated with an index is calculated based on the number of estimated matches 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.

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, accurate index selection or frequent, online index selection.

C. Proposed Solution for Index Selection

Figure 1 shows a flow diagram of the index selection framework. Details about the steps and the dataflow follow:

Initialize Abstract Representations

The initialization step uses a query history file 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 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 history file*. This set is computed using traditional data mining techniques. Considering the attributes involved in each query from the input *query history file* to be a single transaction, *P* consists of the sets of attributes that occur together in a transaction at a ratio greater than the input *support*. So, if the input support level 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. 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 from query optimizer log could just as easily be substituted for the *query history file* in this step.

Query Set Q

An abstract representation of the *query history file* is constructed by including the potential indexes that *could* be beneficial for each 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. The input *histogram size* dictates the number of bits used to represent each unique bucket in the histogram. These bits are designated to represent the single attributes that met the input *support* in the input *query history file*. The number of bits that each of the 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 history file*. 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 go through records, we only aggregate the count of records 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 I 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 histogram records 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 I

HISTOGRAM EXAMPLE

Calculate Query Cost Using Index

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 each of that query's possible indexes. 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 get 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 possible index for each query. 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. We derived a formula to estimate cost for partial match queries for R-trees from the cost model proposed in [5] for expected number of page accesses for range queries for R-trees. The formula we use for multi-dimensional index cost is $(d^{(1/2)} * m^{(1/d)})^d$ where d is the dimensionality of the index and m is the number of matches returned using that index. This function takes advantage of the potential lower number of matches associated with higher dimension indexes while penalizing for the additional overhead caused by the increase in index dimensionality.

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.

Index Selection Loop

After initializing the index selection data structures and updating query costs for each query-index pair, we use a greedy algorithm to iteratively select indexes that would be appropriate for the given query history file and data set. For each index in the potential index set P, we traverse the queries in query set Q 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.

Throughout this loop, where possible, we prune the complexity of the abstract representations

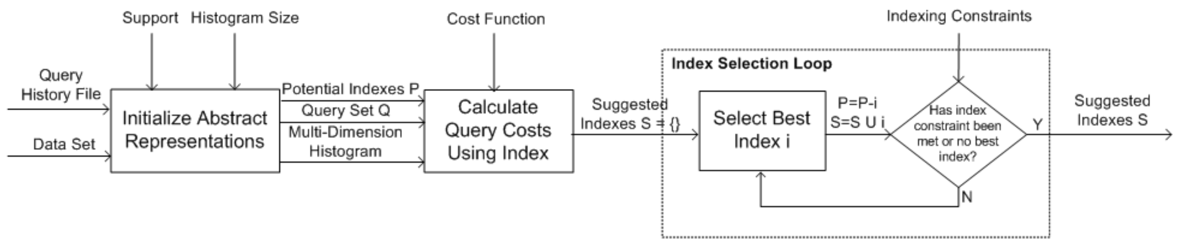

Fig. 1. Index Selection Flowchart

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 not considering 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* level.

D. Proposed Solution for Dynamic Index Selection

We use control feedback to monitor the performance of the current index set for incoming queries and to 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.

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.

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 static index selection algorithm.

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 for the abstract representation of q. We also incrementally compute the attribute sets that meet the input *support* 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.

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 simply 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* 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 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.

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.

Coarse Control Loop

The coarse control loop is used to recommend more costly, but changes with greater impact on future performance to 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

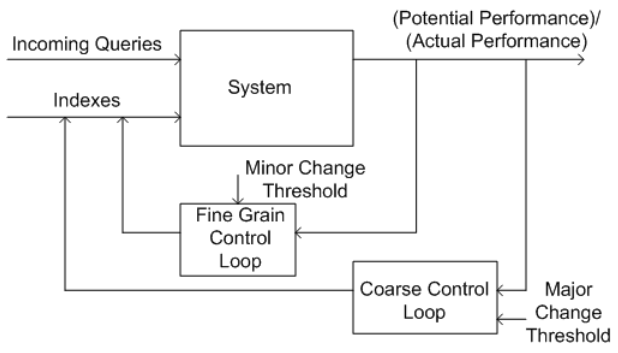

Fig. 2. Dynamic Index Analysis Framework

are made to update the system data structures to the new situation. Increasing the input *major change threshold* increases the frequency of major changes.

IV. Empirical Analysis

A. Experiment Objectives

We had several goals for the experiments. First, we wanted to show that our static index selection algorithm yields reasonable results in terms of the indexes selected and the analysis time. We also wanted to show that the algorithm is flexible and could be tailored based on the constraints of the application, whether the constraints are related to index storage space or computational time. Finally, we wanted to show that the change detection and dynamic index selection algorithm yields consistent performance over query access patterns that evolve over time.

B. Experimental Setup

Data Sets

We used several data sets 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

We analyzed the effect of varying several analysis input parameters including support, multi-dimensional histogram size, and online indexing control feedback decision thresholds.

Query Workloads

At this point, we wish to explore the general behavior of database interactions and do not want to inadvertently observe the coupling associated with using a specific query history on the same database. Therefore, we generated query workload files by merging synthetic query histories and query histories from real-world applications with different data sets. We merge these by taking a random record from our data set and the numbers 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 we randomly selected the n^{th} record from our data set, we would generate a SELECT type query from our 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.

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% {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

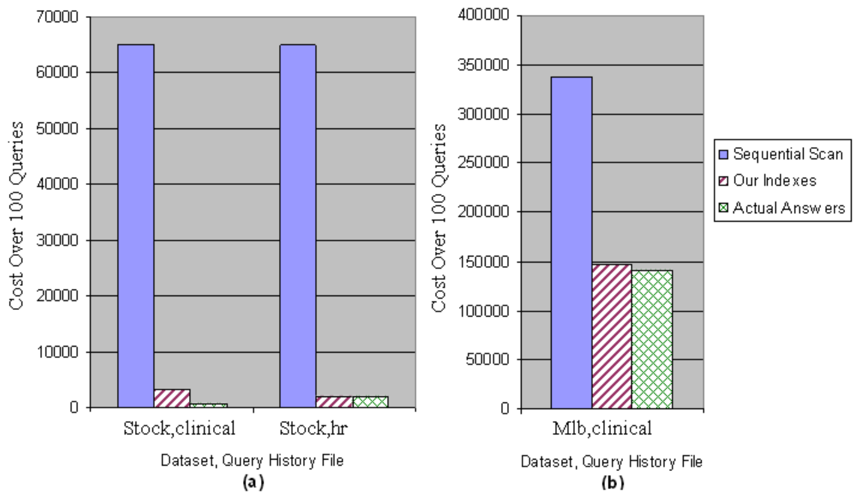

Fig. 3. Costs for Ideal, Sequential Scan and Our Index Selection Technique using Relaxed Constraints

file has 54 attributes. Due to the size of this query set, we only use some initial portion of the queries in some experiments.

C. Experimental Results

1) Index Selection with Relaxed Constraints: Figure 3 shows how accurately our static index selection technique can perform. For this scenario, we have used a low support value, 0.01, have placed no constraints on the number of indexes we can select, and use 256 bits for the multi-dimension histogram. In the graph, we show the cost of performing a sequential scan over 100 queries using the indicated data sets, the estimated cost of using our recommended indexes, and the true number of answers for the set of queries. The cost for sequential scan is discounted to account for the fact that sequential access is significantly cheaper than random access. We used a factor of 10 as the ratio of random access cost to sequential access cost, but 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 II compares our index selection algorithm with relaxed constraints against SQL Server's index selection tool, AutoAdmin [9], using the data sets and query workloads indicated.

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

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

TABLE II

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

the query improvement will be very similar using either recommended index set. Our algorithm executes in less time and generates indexes which are closer to the query patterns in the workload. 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 on 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.

2) Baseline Online Indexing Results: Figures 4 through 7 show a baseline comparison of using the query pattern change detection and modifying the indexes against making no change to the initial index set for a number of data set, query history combinations. 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

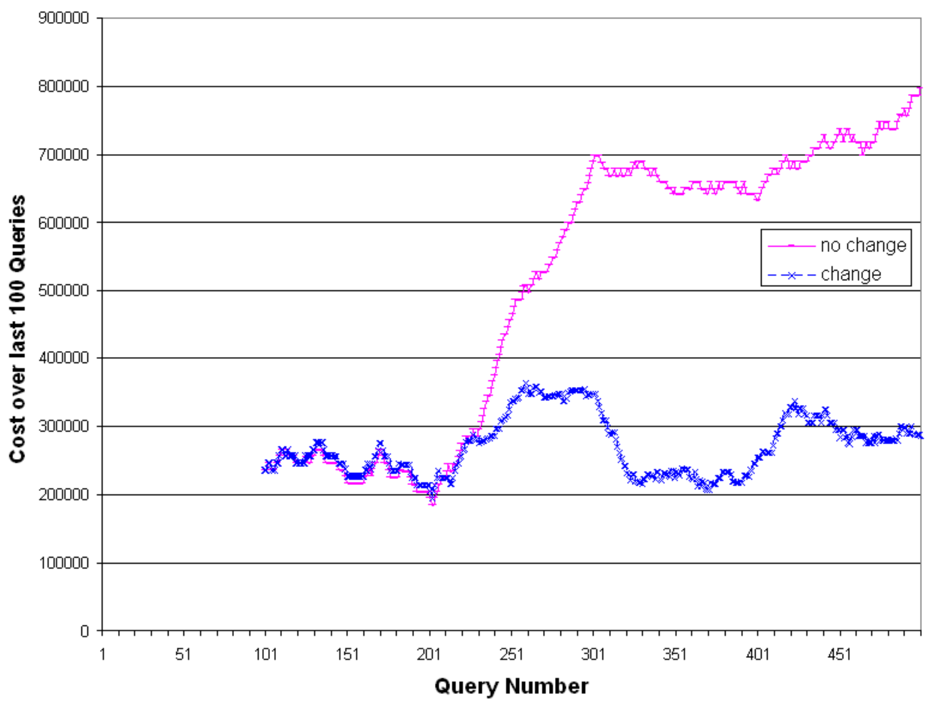

Fig. 4. Baseline Comparative Cost of Online versus Static Indexing, *random* Dataset, *synthetic* Query Workload

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 close to the query patterns 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 online system is better able to absorb the change.

Figure 6 shows the comparison for the *stock* data set using the *synthetic* query workload. It is interesting to note that the shape of this graph and the graph for the *random* data set are nearly identical. This indicates that when the selectivity of the queries is low, the performance of change detection is more a function of the changing query patterns rather than of the data itself.

Figure 7 shows the comparison for the *stock* data set using the first 2000 queries of the *hr* query workload. The online system shows consistently lower cost than the static system and in

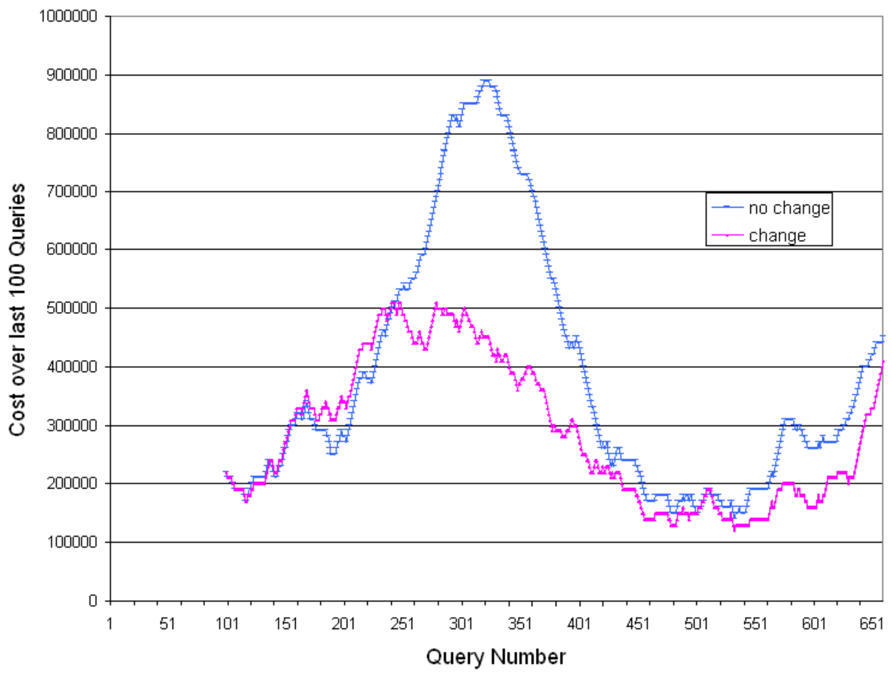

Fig. 5. Baseline Comparative Cost of Online versus Static Indexing, *random* Dataset, *clinical* Query Workload

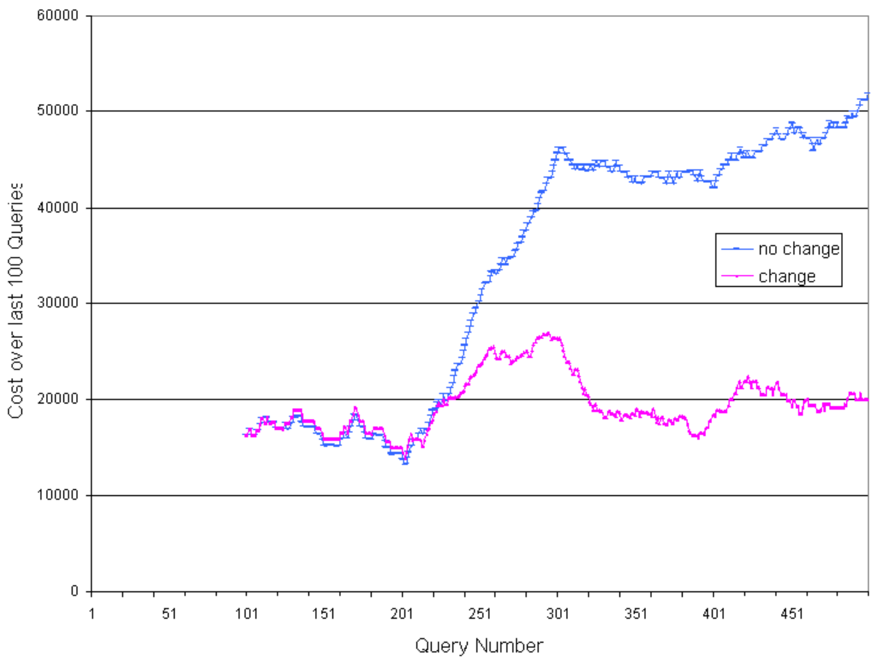

Fig. 6. Baseline Comparative Cost of Online versus Static Indexing, *stock* Dataset, *clinical* Query Workload

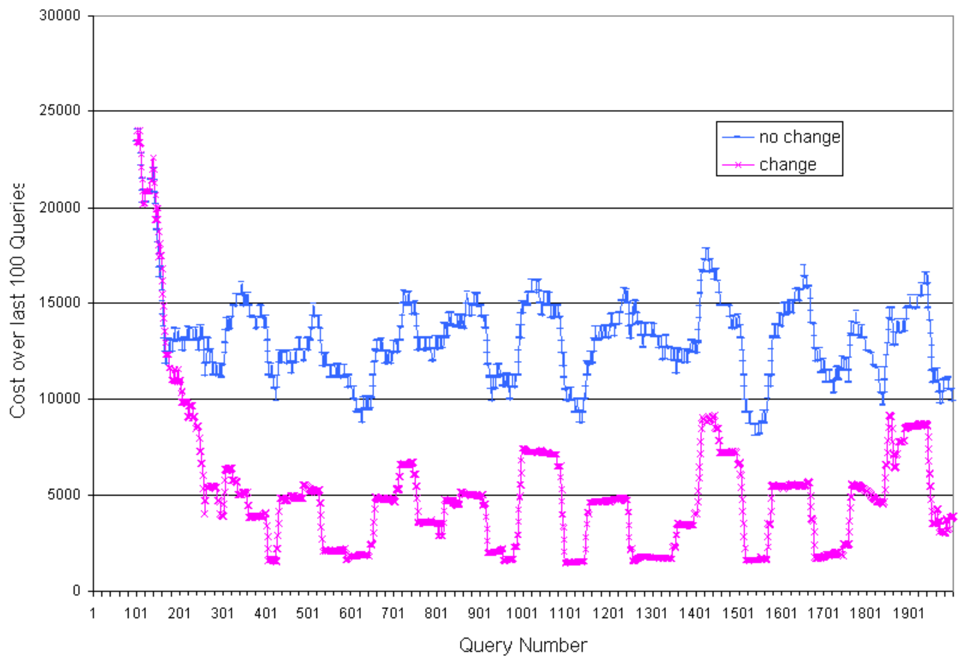

Fig. 7. Baseline Comparative Cost of Online versus Static Indexing, *stock* Dataset, *hr* Query Workload

places significantly lower cost for this real query workload.

3) 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 indexes themselves more complicated and less conducive in an online setting. Figure 8 shows the effect of changing support on the online indexing results for the *random* data set and *synthetic* query workload, while figure 9 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. For the *synthetic* query workload, when the patterns changed, but do not change again, lower support levels translated to better performance. However, for the *hr* query workload, in some areas the 10% support line shows better performance than the 6% and 8% lines. 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

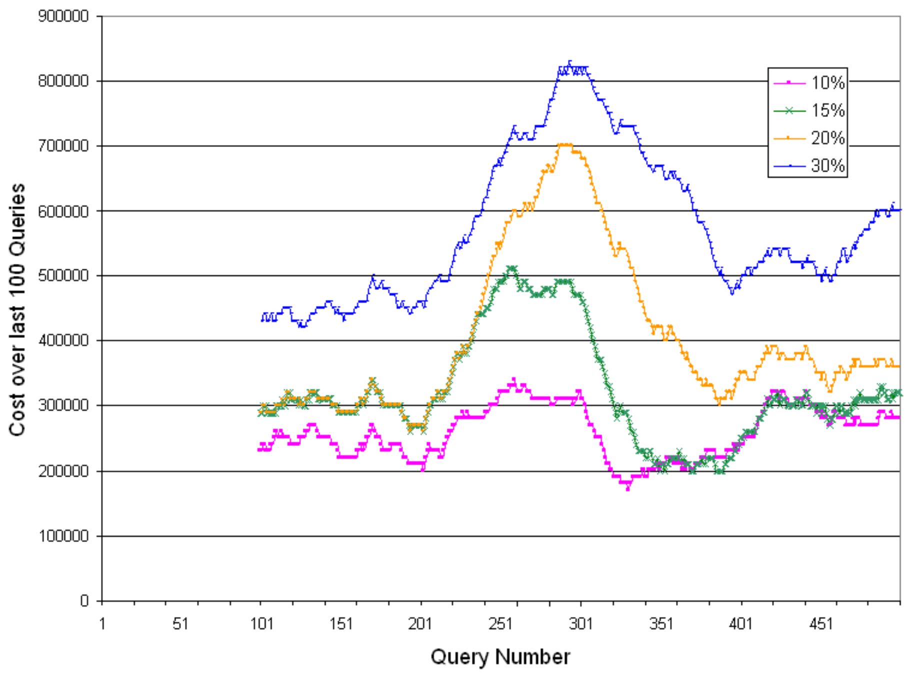

Fig. 8. Comparative Cost of Online Indexing as Support Changes, *random* Dataset, *synthetic* Query Workload

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 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 10 shows the effect of varying the major change threshold in the coarse control loop for the *random* data set and *synthetic* query workload. Figure 11 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

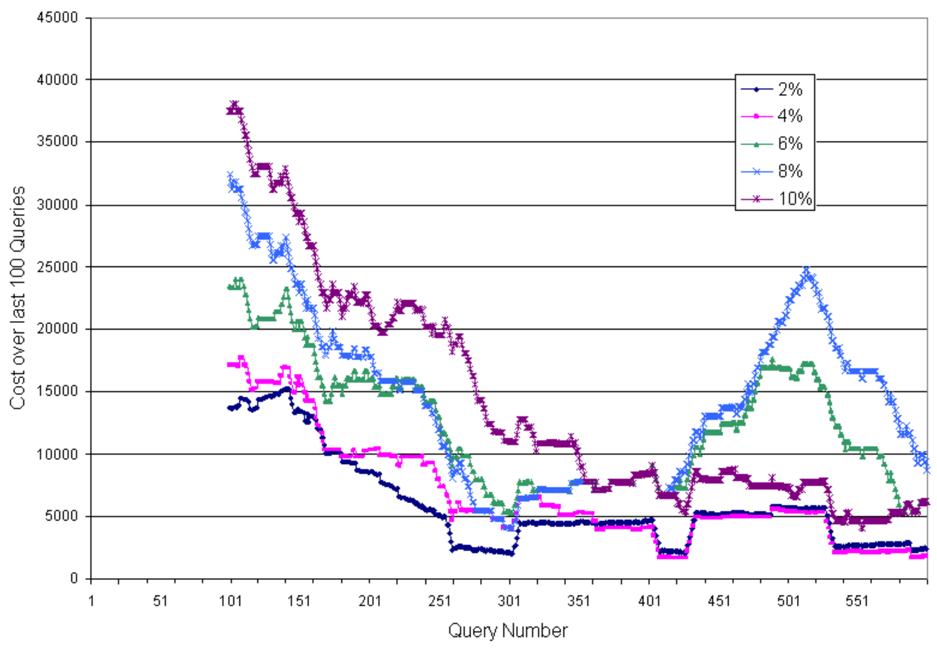

Fig. 9. Comparative Cost of Online Indexing as Support Changes, *stock* Dataset, *hr* Query Workload

graphs show no real benefit once the major change threshold increases (and therefore frequency of major changes) beyond 0.6. Figure 10 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.

Multi – dimensional Histogram Size - increasing the number of bits improves analysis accuracy at the cost of histogram generation time and space. In our experiments we found that for a given analysis setup, there was a number of bits which gave us very close to ideal results, and increasing this value beyond this point had very little impact on the analysis accuracy. Figure 12 shows the effect of varying the multi-dimensional histogram size. We show a much higher cost for using a 32 bit size for each unique histogram bucket, than larger sizes. One reason for this is that we are artificially calculating higher than actual costs because of the lower histogram resolution. We also do not recommend some beneficial index changes because we do not accurately estimate costs in the analysis. As the size of the histogram increases beyond the point where we estimate the number of matches for a query fairly well, we gain very little

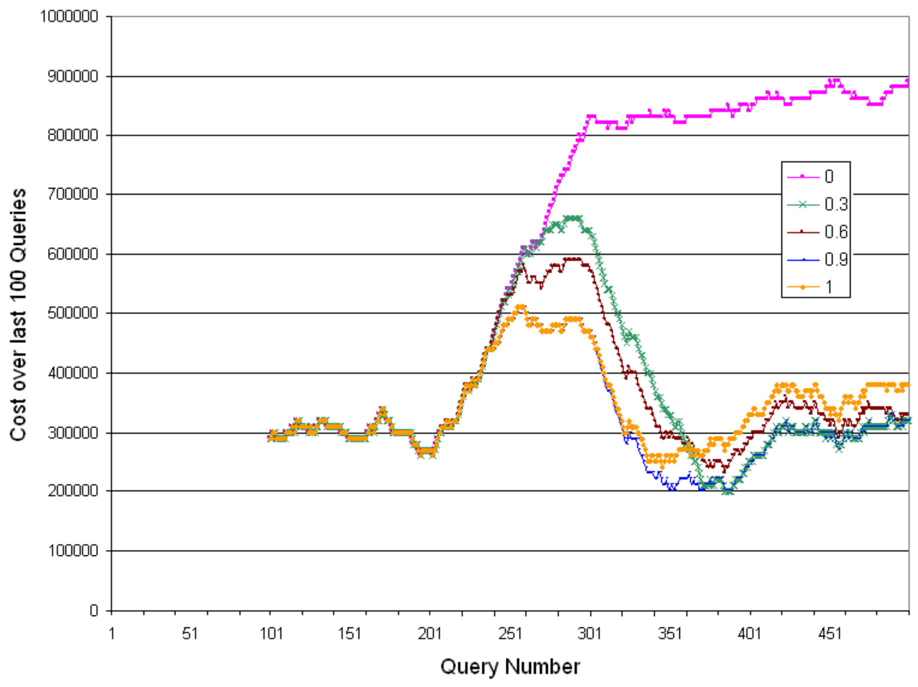

Fig. 10. Comparative Cost of Online Indexing as Major Change Threshold Changes, *random* Dataset, *synthetic* Query Workload

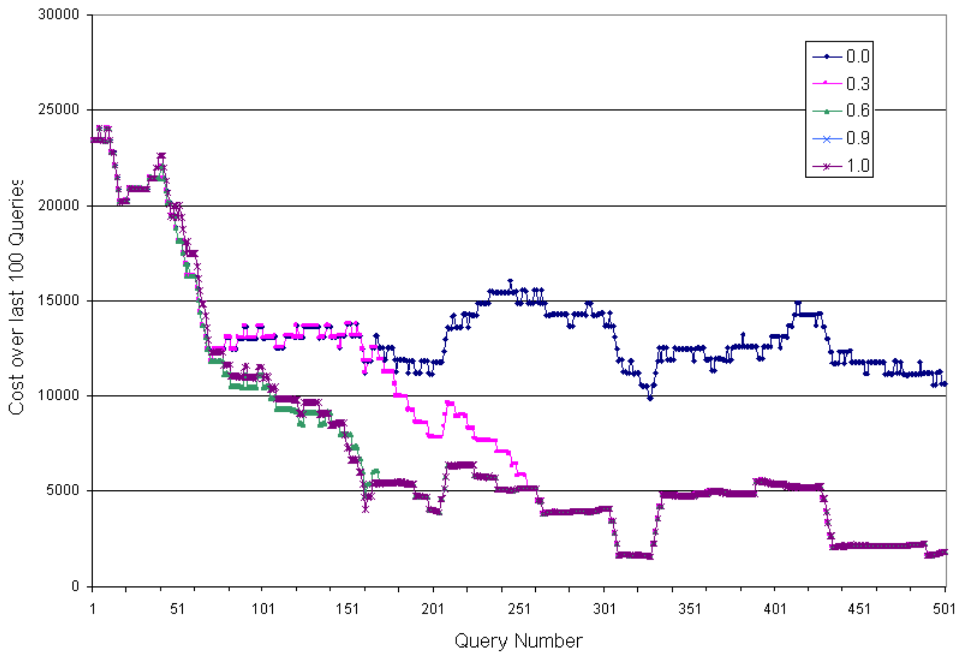

Fig. 11. Comparative Cost of Online Indexing as Major Change Threshold Changes, *stock* Dataset, *hr* Query Workload

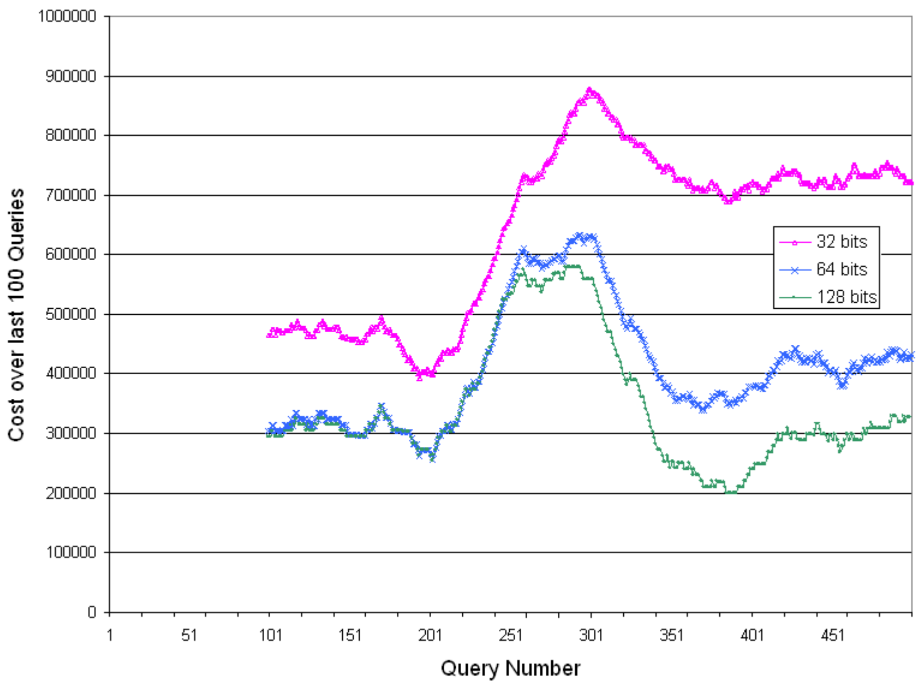

Fig. 12. Comparative Cost of Online Indexing as Multi-Dimensional Histogram Size Changes, *random* Dataset, *synthetic* Query Workload

benefit.

V. Conclusions

We have introduced a flexible technique for index selection 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 higher constraint, less complex analysis is more conducive for online index selection for evolving query patterns. The technique uses our own 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.

These experiments have shown great opportunity for improved performance using adaptive indexing over real query patterns. We introduce a control feedback technique 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, we can tune the analysis time/pattern change recognition tradeoff. The foundation provided here will be used to explore this tradeoff and to develop an improved utility for real-world applications.

Current index selection tools are based on the assumption that the query workload of the future will approximate the query workload of the past. Our technique affords the opportunity to adjust indexes to new query patterns. A limitation of our approach is that if query pattern changes occur more frequently than we make index set changes to respond to them, this control feedback can not be used to consistently affect positive change. However, we can address this by adjusting control sensitivity or by changing control sensitivity over time as we learn more 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, we have affected the frequency of index set change through the use of the online indexing control feedback thresholds. Alternatively, we could adjust the frequency that we monitor the output 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.

We introduced an online change detection system that 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.

REFERENCES

[1] R. Agrawal, C. Faloutsos, and A. Swami. Efficient similarity search in sequence databases. In *4th Int. Conference on Foundations of Data Organization and Algorithms*, pages 69–84, 1993.

[2] 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*, pages 1110–1121, 2004.

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

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

[5] C. Bohm. A cost model for query processing in high dimensional data spaces. *ACM Trans. Database Syst.*, 25(2):129–178, 2000.

[6] 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.

[7] K. R. Castleman. *Digital Image Processing*. Prentice-Hall, Inc., 1996.

[8] K. Chakrabarti, M. Garofalakis, R. Rastogi, and K. Shim. Approximate query processing using wavelets. In *26th Intl. Conf. on Very Large Data Bases*, September 2000.

[9] S. Chaudhuri and V. Narasayya. Autoadmin 'what-if' index analysis utility. In *Proceedings ACM SIG-MOD Conference*, pages 367–378, 1998.

[10] S. Chaudhuri and V. R. Narasayya. An efficient cost-driven index selection tool for microsoft SQL server. In *The VLDB Journal*, pages 146–155, 1997.

[11] D. W.-L. Cheung, S. D. Lee, and B. Kao. A general incremental technique for maintaining discovered association rules. In *Database Systems for Advanced Applications*, pages 185–194, 1997.

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

[13] 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.

[14] A. Dogac, A. Y. Erisik, and A. Ikinci. An automated index selection tool for oracle7: Maestro 7. Technical report, TUBITAK Software Research and Development Center, 1994.

[15] C. Faloutsos, R. Barber, M. Flickner, J. Hafner, W. Niblack, D. Petkovic, and W. Equitz. Efficient and effective querying by image content. *Journal of Intelligent Information Systems*, 3:231–262, 1994.

[16] 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.

[17] V. Ganti, J. Gehrke, and R. Ramakrishnan. Demon: Mining and monitoring evolving data. In *International Conference on Data Engineering*, pages 439–448, San Diego, USA, 2000.

[18] M. Garofalakis and P. Gibbons. Wavelet synopses with error guarantees. *Sigmod*, 2002.

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

[20] B. Jawerth and W. Sweldens. An overview of wavelet based multiresolution analysis. In *SIAM Review*, 1994.

[21] N. Jayant and P. Noll. *Digital Coding of Waveforms*. Prentice-Hall, Inc., 1984.

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

[23] T. Kailath. *Modern Signal Processing*. Springer Verlag, 1985.

[24] K. V. R. Kanth, D. Agrawal, and A. Singh. Dimensionality reduction for similarity searching in dynamic databases. pages 166–176, Seattle, Washington, June 1998.

[25] H. Karhunen. Uber lineare methoden in der wahrscheinlich-keitsrechnung. *Ann. Acad. Science Fenn*, 1947.

[26] E. J. Keogh, K. Chakrabarti, S. Mehrotra, and M. J. Pazzani. Locally adaptive dimensionality reduction for indexing large time series databases. pages 151–162, Santa Barbara, CA, 2001. ACM.

[27] J.-H. Lee, D.-H. Kim, and C.-W. Chung. Multi-dimensional selectivity estimation using compressed histogram information. In *Proc. of the ACM SIGMOD International Conference on Management of Data*, pages 205–214, 1999.

[28] M. Loeve. Fonctions aleatoires de seconde ordre. *Processus Stochastiques et Mouvement Brownien*, 1948.

[29] Y. Matias, J. Vitter, and M. Wang. Wavelet based histograms for selectivity estimation. In *ACM Sigmod Intl. Conf. on Management of Data*, June 1998.

[30] Y. Matias, J. Vitter, and M. Wang. Dynamic maintenance of wavelet-based histograms. In *VLDB*, 2000.

[31] W. Niblack, R. Barber, W. Equitz, M. Flickner, E. Glasman, D. Petkovic, and P. Yanker. The QBIC project: Querying images by content using color, texture and shape. In *Proc. of the SPIE Conf. 1908 on Storage and Retrieval for Image and Video Databases*, volume 1908, pages 173–187, Feb. 1993.

[32] A. V. Oppenheim and R. W. Schafer. *Discrete-Time Signal Processing*. Prentice-Hall, Inc., 1989.

[33] S. Papadimitriou, A. Brockwell, and C. Faloutsos. Adaptive hands-off stream mining. In *VLDB*, pages 560–571, Berlin, Germany, Sept. 2003.

[34] 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.

[35] D. Rafiei and A. Mendelzon. Similarity-based queries for time series data. pages 13–25, 1997.

[36] D. Rafiei and A. O. Mendelzon. Efficient retrieval of similar time sequences using DFT. In *Proceedings of the International Conference on Foundations of Data Organizations and Algorithms (FODO)*, Nov. 1998.

[37] K. Rao and J. Hwang. *Techniques and Standards for Image, Video and Audio Coding*. Prentice Hall, Upper Saddle River, NJ, 1996.

[38] K. Rao and P. Yip. *The Transform and Data Compression Handbook*. CRC Press, 2001.

[39] 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.

[40] E. Stollnitz, T. DeRose, and D. Salesin. *Wavelets for Computer Graphics*. Morgan Kaufmann, San Francisco, 1996.

[41] S. Thomas, S. Bodagala, K. Alsabti, and S. Ranka. An efficient algorithm for the incremental updation of association rules in large databases. In *Knowledge Discovery and Data Mining*, pages 263–266, 1997.

[42] 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.

[43] A. Veloso, W. Meira, Jr., M. de Carvalho, B. Possas, S. Parthasarathy, and M. J. Zaki. Mining frequent itemsets in evolving

databases. In *Proceedings of the 2nd SIAM International Conference on Data Mining*, Arlington, Virginia, USA, April 2002.

[44] J. Vitter and M. Wang. Approximate computation of multidimensional aggregates of sparse data using wavelets. In *ACM SIGMOD International Conference on Management of Data*, May 1999.

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

[46] D. Wu, D. Agrawal, A. E. Abbadi, and T. R. Smith. Efficient retrieval for browsing large image databases. pages 11–18, Nov. 1996.

[47] Y. Wu, D. Agrawal, and A. El Abbadi. A comparison of DFT and DWT based similarity search in time-series databases. In *Proceedings of the 9th International conference on Information and knowledge management*, pages 488 – 495, McLean, Virginia, 2000.