Improving Index Maintenance Using Decision Analysis

Pedro Holanda[1], João Paulo Pordeus[1], José Maria Monteiro[1], Angelo Brayner[2], Sérgio Lifschitz[3]

[1] Departamento de Computação, Universidade Federal do Ceará (UFC) - Brasil
{pedroholanda, jpaulo, monteiro}@lia.ufc.br
[2] Programa de Pós-Graduação em Informática Aplicada, Universidade de Fortaleza (UNIFOR) - Brasil
brayner@unifor.br
[3] Departamento de Informática, PUC-Rio - Brasil
sergio@inf.puc-rio.br

Abstract. One of the main activities of a Database Administrator (DBA) is to maintain an appropriate index configuration. However, adjustments in index configuration have become a very complex decision problem for DBAs. Making decisions concerning index maintenance often strains human cognitive capabilities. The DBA is faced with an intractable amount of factors that can impact the result of his/her decision. In order to address this issue a Decision Support System (DSS) can be used to support DBA decisions based on rational and systematic procedures. A DSS can model the chances of success of each DBA's adjustment and the risk acceptance level of each decision maker using a probabilistic framework. This article presents a Decision Support Mechanism for Index Maintenance (DSIM), whose main feature is to provide a DSS to the problem of choosing the best point in time to maintain indexes for quasi-periodic time-consuming SQL queries. DSIM provides a risk acceptance configuration feature using a probabilistic framework. Experiments show that DSIM can be effectively deployed to support DBA's decisions.

Categories and Subject Descriptors: H.2 [**Database Management**]: Miscellaneous; H.3 [**Information Storage and Retrieval**]: Miscellaneous; I.7 [**Document and Text Processing**]: Miscellaneous

Keywords: index maintenance, decision analysis, prediction models, decision under uncertainty

1. INTRODUCTION

Nowadays, database applications may be very complex, dealing with a huge volume of data and a high demand concerning availability, query response time and transaction throughput. In this context, index structures play a fundamental role in improving database performance [Medeiros et al. 2012]. This is because the existence of suitable indexes speeds up the execution of queries submitted to DBMS (Database Management System) [Morelli et al. 2009]. Thus, one of the main activities of a Database Sdministrator (DBA) is to maintain an appropriate index configuration in order to minimize the workload (set of queries and updates submitted to the DBMS) response time. However, manual adjustments in index configuration (creation, drop and reindex) have become very complex activity for current database applications. They require a thorough knowledge of DBMSs implementation aspects, hardware capacity, physical database design strategies, characteristics of the stored data and complex workload [Chaudhuri and Narasayya 2007].

Several tools (e.g., wizards and advisers) and approaches have been proposed aiming at solving the problem of automatic index selection. Examples of such tools include DB2 Advisor [Valentin et al. 2000], Database Tuning Advisor (MS SQL Server) [Agrawal et al. 2004] and SQL Adjust Advisor (Oracle) [Dageville et al. 2004]. Over the last years, some prototypes for automatic and continuous (online) index tuning have been proposed [Bruno and Chaudhuri 2007a; 2007b; Schnaitter et al. 2006; 2007; Luhring et al. 2007; Sattler et al. 2003; 2003; Sattler et al. 2004]. However, most of them are reactive, since they analyze a previously executed workload and infer the best index configuration *after* that workload has been executed. Therefore, reactive approaches react *after* performance problems are detected. For that reason, more recently, proactive solutions have being investigated. Medeiros et al. [2012] present a mechanism, denoted PIM (Proactive Index Maintenance), which is able to predict when a time-consuming query q will be executed, in order to proactively create index structures which reduce its response time. In other words, indexes are automatically created and dropped by PIM in a proactive manner.

However, making decisions concerning index maintenance often strains human cognitive capabilities. The decision maker, DBA, is faced with an intractable amount of factors that can impact the result of his decision. Besides, creating indexes too early (w.r.t. the moment the queries which may use those indexes are executed) increases index maintanance cost, for the index is created with an unnecessary anticipation. On the other hand, planning to create indexes belatedly incurs on the risk of performing the query without the appropriate indexes. In this scenario, it is important to note that the risk acceptance level of the decision maker is an import variable on this problem. The risk acceptance level (RAL) is defined as the probability of executing a query without the corresponding index on a given date. In order to address this issue a Decision Support System (DSS) can be used to uphold DBA decisions based on rational and systematic procedures. A DSS can model the chances of success of each DBA's adjustment and the risk acceptance of each decision maker using a probabilistic framework.

This article presents a Decision Support Mechanism for Index Maintenance (denoted DSIM, for short). The main objective of DSIM is to provide a DSS to the problem of choosing the best point in time to create indexes for quasi-periodic time-consuming SQL queries. Quasiperiodicity is defined as a periodic behavior with a component of uncertainty. Furthermore, DSIM provides a risk acceptance configuration feature using a probabilistic framework and uses different prediction models. The proposed approach relies on the premise that a workload submitted to the DBMS in the future is similar to an already captured one. It is important to note that previous works do not encompass the decision maker (DBA) risk acceptance level.

In order to achieve its goal, DSIM implements the following five steps:

(1) Capture of workload. The database workload is captured and stored in the DSIM metabase, denoted local metabase.
(2) Identification of time-consuming queries and appropriate index structures to execute such queries. DSIM uses the strategy presented by Medeiros et al. [2012], which analyzes the previous captured workload to identify time-consuming queries and the most adequate index structures to each query.
(3) Selection of the most appropriate prediction model (for example, Linear Regression, Multi-Layer Perceptron - MLP, Radial Basis Function - RBF, etc) and the best configuration (values of required parameters for the chosen model), for each time-consuming query.
(4) Uncertainty estimation. The uncertainty is quantified using a probabilistic framework. DSIM does not forecast the moment when a specific time-consuming query will run again, but a full probability distribution (PD), represented by a prediction histogram, for the day that this query will be submitted next time. For this, the bootstrap method is used.
(5) Decision support. Based on the prediction histogram, provided on the previous step, and the risk acceptance level (RAL) of the decision maker, gives as input parameter, DSIM will indicates the ideal instant to create the necessary indexes for a specific query.

The proposed approach has been implemented and its efficiency has been evaluated in a real environment. The experimental results show the feasibility of predicting the moment in which time-consuming queries are executed according to a given risk acceptance level. Furthermore, the results also indicate that the proactive index creation under uncertainty for time-consuming queries provides significant performance gains, even if the created indexes are dropped shortly after queries execution. In DSIM, two prediction models have been implemented. One model is based on linear regression and the other one implements in fact two different types of neural networks: multilayer perceptron (MLP) and radial basis function (RBF). Those models were chosen because they are widely used in time series forecasting. Nonetheless, others models can be added to the proposed mechanism.

The remainder of this article is organized as follows. Section 2 describes approaches related to the index maintenance problem. Section 3 discusses the key concepts utilized to describe the proposed approach. In turn, Section 4 presents the proposed approach to support decision-making activities about index maintenance. Section 5 describes the experimental results. Finally, Section 6 concludes this article and points out directions for future research.

2. RELATED WORK

In this section we present and discuss some relevant approaches for index tuning.

Bruno and Chaudhuri [2007a] and Bruno and Chaudhuri [2007b] present an intrusive index tuning tool implemented as a component of Microsoft SQL Server 2005. This tool runs continuously and, either reacting to variations in the workload or considering data characteristics, it modifies the database physical design. The tool proposed by Bruno and Chaudhuri [2007a] and Bruno and Chaudhuri [2007b] works as follows: during the optimization process for a given query q, the optimizer call is diverted to an *Index Analysis* (IA) module, which identifies a set of candidate indexes CI that could potentially reduce the execution time for q. For this purpose, IA uses AND/OR request trees and local transformations. Thereafter the query q is optimized and processed as usual. During the execution of q, IA estimates the potential benefits if candidate indexes were used to process q. Moreover, it evaluates the benefits of real indexes, which were used during query execution. These steps are performed by the *Cost/Benefit Adjustment* (CBA) module. After a query q is executed, IA is triggered once more to analyze the cost/benefit of using candidate indexes belonging to CI. Based on this analysis, the IA module sends requests for creating or dropping indexes to the *Asynchronous Task Manager* module (ATM). It is important to note that this approach adds some new tasks that are executed before and during query optimization. Consequently, such a feature may negatively impact the query processing performance.

Schnaitter et al. [2006] and Schnaitter et al. [2007] present a prototype of a self-tuning framework called COLT (Continuous On-Line Tuning). This tool continuously monitors queries submitted to the DBMS and autonomously adjusts the index configuration, considering space restrictions for these structures. COLT was implemented in an intrusive manner into PostgreSQL, since it replaces the PostgreSQL's query optimizer by a module, denoted *Extended Query Optimizer* (EQO), and adds a Self-Tuning Module (STM) to PostgreSQL's query engine. COLT implements the classic self-tuning steps composed by Observation, Prediction and Reaction phases. In order to achieve its goal, for each submitted query q the STM selects a set of candidate indexes (CI), which includes hypothetical and real indexes, and sends CI to EQO. For each index $i \in CI$, EQO generates a new query plan for q that uses the index i and computes the corresponding gain. Thus, if CI has n indexes, EQO has n query plans for q. In COLT, the duration of the observation phase is defined by the notion of an *epoch*. More specifically, an *epoch* represents the time for executing ten queries. During an *epoch* the STM gathers statistics about the workload and computes the gains of using candidate indexes. At the end of an *epoch*, the set of candidate indexes (materialized or hypothetical) is analyzed to verify the need for changes in the index configuration. In this case, hypothetical indexes may be materialized and materialized indexes may be removed (in this case, they become hypothetical indexes). The main drawback in this COLT approach stems from the notion of *epoch*. By assigning to an *epoch* a fixed number of 10 queries, COLT may take too much time to react. Actually, any fixed number may not be consistent with the database system needs.

Luhring et al. [2007] propose extensions to PostgreSQL for supporting index self-tuning functions. That approach is intrusive and uses the classic self-tuning strategy composed of the Observation, Prediction and Reaction phases. These steps are performed sequentially and continuously. During the observation phase, each submitted query q is analyzed in order to discover candidate indexes and the heuristic described by Valentin et al. [2000] is used. Thereafter, q is optimized twice, once without considering any index and another considering all candidate indexes. This approach also applies the notion of an *epoch* [Sattler et al. 2004] to define the duration of the observation phase. However, Luhring et al. [2007] defines an epoch in terms of the maximum number of recommendations for the same index.

Sattler et al. [2003] and Sattler et al. [2004] propose a middleware for IBM DB2 that automatically suggests the creation of indexes. That solution is based on DB2 proprietary commands (e.g *SET CURRENT EXPLAIN MODE RECOMMEND INDEXES*), which are not available for other DBMSs. Furthermore, it requires all SQL clauses to be forwarded to the middleware, and not to the DBMS. Their approach presents two critical drawbacks: (i) code rewriting becomes mandatory for previously existing applications and (ii) workloads directly submitted to the DBMS cannot be managed since the workload is forwarded to the middleware.

The work proposed by Morelli *et al* [Salles and Lifschitz 2005; de Carvalho Costa et al. 2005; Morelli et al. 2009] presents a self-tuning component implemented within the PostgreSQL code, allowing autonomous index creation, dropping and rebuilding. The optimizer takes into account hypothetical indexes for the construction of alternative query plans. The developed prototype does not consider important restrictions such as those regarding the physical space available for the materialization of the suggested (hypothetical) indexes.

Maier et al. [2010] and Alagiannis et al. [2010] presents an intrusive solution, called PARINDA (PARtition and INDex Advisor). It is an interactive physical designer for PostgreSQL that works as follows: given a workload containing a set of queries, PARINDA allows the DBA to manually suggest a set of candidate indexes and the tool shows the benefits of the suggested index set and index interactions visually. Besides, the tool can find the optimal index partition for a given query workload. It also suggests a schedule to implement the suggested indexes. Finally, the tool can continuously monitors the performance of the DBMS under incoming queries, and it suggests new indexes when they offer sufficient workload speedup. Bruno and Chaudhuri [2010] propose an intrusive and interactive tool for Microsoft's SQL Server, which is similar to PARINDA since it makes tuning sessions interactive, allowing DBAs to try different tuning options and interactively obtain a feedback.

Medeiros et al. [2012] present a mechanism, denoted Proactive Index Maintenance (PIM, for short), for proactive index management based on the use of prediction models. The main objective of the proposed mechanism is to predict when a time-consuming query q will be executed, in order to proactively create index structures that reduce $q's$ response time. Thus, indexes are automatically created and dropped by PIM in a proactive manner. Experiments show that PIM presents low overhead, can be effectively deployed to predict time-consuming query execution and provides significant performance gain during time-consuming query execution. Different prediction models have been evaluated: neural networks (Multi-Layer Perceptron - MLP and Radial Basis Function - RBF) and Linear Regression. The results indicate that the prediction model is query-specific, i.e., it should be defined according to the statistical distribution (normal, poisson, binomial) of the query execution history.

This article proposes a Decision Support Mechanism to help DBAs to solve the problem of choosing the best instant to maintain indexes for quasi-periodic time-consuming SQL queries. For each time-consuming query q, DSIM retrieves from the Local Metabase the execution history of q, denoted H_q. Based on H_q DSIM infers the most appropriate prediction model and the best prediction instance pi_{i_j} for q. However, differently from the approach of Medeiros et al. [2012], the selected prediction instance pi_{i_j} is not used to forecast the moment when q will run again. DSIM delivers a full probability distribution, represented by a prediction histogram, for the day that q will be submitted next time. Based on the prediction histogram and the RAL, DSIM indicates the optimal instant to create the necessary indexes for q. It is important to note that previous works do not encompass the decision maker RAL. For that reason, DSIM is an interactive Decision Support System for Index Maintenance, since it allows DBAs to try different risk acceptance levels and interactively obtain a feedback. Similar to PIM [Medeiros et al. 2012], DSIM is a non-intrusive solution.

It is important to emphasize that DSIM can not be classified as a reactive or proactive solution. However, if the DBA sets a risk acceptance level, DSIM can be configured to operate in a semi-automatic manner, generating alerts, but leaving the decisions for DBAs, or in a automatic manner, together with PIM, sending to PIM the ideal instant to create the necessary indexes for q according to the defined RAL.

3. BASIC CONCEPTS

This section presents the key concepts on which the proposed approach is based.

3.1 Decision under uncertainty

Making decisions for solving complex problems often strains human cognitive capabilities. In many real world problems, decision makers face an intractable amount of factors that can impact the result of their decisions. In order to address this issue, the interest in Decision Support Systems (DSS) is growing rapidly [Drudzdel and Flynn 2002]. DSS can be defined as methods or tools to support decisions based on rational and systematic procedures. Applications of this concept can be found in many areas such as engineering [Dahala et al. 2014], medicine [Bourouisa et al. 2014] and business [Kima et al. 2014]. The great amount impacting factors present in these real applications can be grouped by the concept of uncertainty.

Decision under uncertain environments is a challenging task that is addressed in the specialized literature mainly by the usage of statistics and probability theory. The influence of many different factors may be well represented by probabilities and probability distributions. For example, a marketing manager may decide among a variety of marketing strategies for a new product knowing that the success of the chosen strategy depends strongly of many factors, and some of them are unknown. Based on previous experiences, a DSS designer can model the chances of success of each strategy, using a probabilistic framework. This model can then be used to support the marketing strategy choice [Bertsimas and Freund 2004].

One important feature of using this probabilistic framework is the possibility of modeling the risk acceptance of each decision maker. It is well known that different persons have different risk acceptance levels and this can have a great impact on the success of a DSS. It is desirable that the decision maker feel comfortable with the DSS recommended strategy. Work related to this topic can be found in [Kima et al. 2014], [Bertsimas and Freund 2004] and [Dahala et al. 2014].

The present work proposes a DSS to the problem of choosing the best instant to design indexes for quasi-periodic database queries. The use of these indexes can reduce significantly the time spent for each query. The proposed framework provides a risk acceptance configuration feature using a probabilistic framework.

3.2 Time-consuming queries

Intuitively, a time-consuming query has high response time whenever it is executed and it is not executed in regular time intervals, but it is executed very often.The solution proposed to identify the time-consuming queries is based on the concept of benefit, initially proposed by Salles and Lifschitz [2005]. Intuitively, the notion of benefit quantifies the gain in using of a specific index structure i to process a query q.

Definition 3.1 *Benefit.* Let $B_{i,q}$ be the benefit provided by the index structure i for processing a query q. The benefit of using i to run q is computed as follows:

$$B_{i,q} = max\{0, cost(q) - cost(q,i)\},$$

where, $cost(q)$ represents the cost of running the query q without the use of index i and $cost(q, i)$ represents the cost of running the query q using the index i.

Now we can formally define the concept of time-consuming queries.

Definition 3.2 *Time-Consuming Query.* A query q is considered "time-consuming" if and only if: *(i)* $RT_q > t$, where RT_q is the response time of q and t is a constant (parameter), *(ii)* $F_q < k$, where F_q is the number of executions of q divided by the size of the observation period, given in months, and k is a constant (parameter), and *(iii)* there is at least one index structure i such that $B_{i,q} > EC_{C_i}$, where EC_{C_i} is the estimated cost of creating the index structure i.

4. PROPOSED METHOD

In this section we describe the proposed approach to support decision-making activities about index maintenance, denoted DSIM (Decision Support Mechanism for Index Maintenance). The approach proposed in this work to support index maintenance consists of five steps, detailed next.

4.1 Step 1: Capturing the workload

This step consists of extracting from the target DBMS's log the workload submitted to the DBMS. For this, the DBA may check the DBMS's log and provide a file containing the database workload. For that, the DBA may use the tool presented by Medeiros et al. [2012], denoted PIM. This tool access the DBMS catalog in order to get the tasks executed by the DBMS.

Of course, the way to obtain database worloads varies from one DBMS to another, since each DBMS has its own catalog format. Thus, to encapsulate such differences PIM implements the mechanism of drivers. In this sense, there is a specific driver for each DBMS.

Note that, by employing the notion of drivers, PIM implementation becomes independent of particular aspects of each DBMS catalog. A database workload is a set of tasks submitted to DBMS in a given period of time. A task is a triple containing <*SQL Expression, execution plan, estimated cost*>. The database workload is stored in a structure, called Local Metabase. Therefore, not only queries but their corresponding execution plans and execution costs are stored and become available to support index tuning activity.

4.2 Step 2: Identifying time-consuming queries and efficient index structures

In order to identify time-consuming queries and the most appropriate indexes for each one of these queries, two strategies are possible: *(i)* the DBA may provide to DSIM the set of time-consuming queries and the most appropriate indexes for each query and *(ii)* the DBA can use the mechanism presented by Medeiros et al. [2012], denoted PIM. The mechanism proposed by Medeiros et al. [2012] uses the concept of Hypothetical Execution Plan (for short, HP).

The key idea behind the concept of HP is to identify indexes which could bring benefits to a given query. Thus, after obtaining the real execution plan of query q (i.e., the execution plan generated by the DBMS native query optimizer), the captured real plan is traversed for searching for operations, which do not use indexes, such as full scans on tables. The goal is to replace such operations in hypothetical execution plans by equivalent physical operations that make use of indexes, such as index scans. Hypothetical execution plans may have real or hypothetical indexes.

A real index structure physically exists in disk, while a hypothetical index structure does only exist in the metabase. Indexes identified during the execution of this step are considered the most appropriate to accelerate the execution of the analyzed query. PIM selects hypothetical indexes whose benefit to the analyzed query q ($B_{i,q}$) is greater than its creation cost (EC_{C_i}). The set of indexes

Algorithm 1 DSIM_S Heuristic: Instance Prediction Selection

```
 1: for each captured query $q \in LM$ do
 2:     $H_q \leftarrow$ the execution history of $q$
 3:     $best_prediction_instance_q \leftarrow \emptyset$
 4:     $smaller_RMSE_q \leftarrow 1$
 5:     for each prediction model $pm_i \in pm$ do
 6:         for each prediction instance $pi_{i_j} \in pm_i$ do
 7:             build $pi_{i_j}$ using $H_q$
 8:             RMSE $\leftarrow$ the RMSE for $pi_{i_j}$
 9:             if RMSE $< smaller_RMSE_q$ then
10:                 $smaller_RMSE_q \leftarrow$ RMSE
11:                 $best_prediction_instance_q \leftarrow pi_{i_j}$
12:             end if
13:         end for
14:     end for
15: end for
```

Fig. 1. DSIM_S Heuristic: Prediction Instance Selection

selected for a query q is called I_q. The set of time-consuming queries is automatically built by PIM as follows. For each SQL query q stored in the metabase, PIM retrieves $q's$ real execution plan (generated by the native query optimizer), which is stored in the metabase as well. Thus, PIM is able to construct a hypothetical plan of q. By using Definition 3.2, PIM classifies q as a time-consuming or not. Next, using the information provided by DBA or PIM, for each time-consuming query q, DSIM stores in the Local Metabase: *(i)* the information that q is a time-consuming query and *(ii)* the set of most appropriate indexes for q, denoted I_q.

4.3 Step 3: Selecting appropriate prediction models for time-consuming queries

In this step, initially, for each time-consuming query q, DSIM retrieves from the Local Metabase the execution history of q, denoted H_q. The execution history consists on a vector containing the last n executions of the query q. Next, based on H_q DSIM will select the most appropriate prediction model (for example, Linear Regression, Multi-Layer Perceptron - MLP, Radial Basis Function - RBF, etc) and the best prediction instance for q. For this, the following strategy is used. Be $pm = pm_1, pm_2, \cdots, pm_k$ a set of prediction models and $pi_i = pi_{i_1}, pi_{i_2}, \cdots, pi_{i_l}$ a set of prediction instance for the prediction model pm_i, where $1 \leq i \leq k$, given as input. A prediction instance pi_{i_j} is a valid configuration (values of required parameters) for the prediction model pm_i. DSIM will evaluate each prediction instance to find out which one gives the best results, regarding the metric root mean squared error (RMSE). The prediction instance with smaller RMSE is associated to q. However, different from the work of Medeiros et al. [2012], the selected prediction instance pi_{i_j} will not be used to predict the next executions of q. The prediction instance pi_{i_j} will be used as input in the next step in order to build a prediction histogram.

Figure 1 shows the algorithm used to select the most appropriate prediction model and the best prediction instance for each time-consuming query.

4.4 Step 4: Uncertainty estimation

In DSIM, the uncertainty is quantified using a probabilistic framework. In this framework, not only a time instant but a full distribution is predicted. For instance, given a vector containing the last days that a query q was executed, the proposed method provides, as an output, a probability distribution (PD) for the day that q will happen next time.

As the prediction methods presented in Section 4.3 are only capable of predicting a single point, given an input vector $\mathbf{v}_q$, it is necessary to apply one strategy to estimate a probability distribution (PD) given single point estimation methods. For choosing a distribution estimation method for decision-making support about index maintenance, the method should be able to work with different prediction models (e.g., Linear Regression, RBF, MLP, etc). This constraint can be satisfied by using the bootstrap uncertainty estimation technique [Efron 1979]. The bootstrap technique allows the approximation of the probability distribution by an histogram using a simple resample scheme.

The procedure to run the bootstrap technique is described in the following steps:

(1) From the original dataset (H_q), obtain B resamples of units based on a sampling with replacement scheme.
(2) From each B resamples, build a new prediction instance using the prediction model (Linear Regression, RBF, MLP, etc) and the configuration selected in the previous step. This will result on B new prediction instances.
(3) For a given query historical data (H_q), predict the next execution of query q using all the B prediction instances.
(4) Build an histogram from the results.

In order to illustrate the aforementioned procedure, consider a dataset containing seven executions of a given query q. The executions happened on days 50, 61, 69, 80, 93, 101 and 110. The prediction task is defined by predicting the future query execution instant based on the last k execution points in time. Now, let us build the dataset using the last three query execution instants, i.e., 93, 101 and 110. In order to predict this instants DSIM has as input the last $k = 4$ points in time in which the query q has been executed before each of the three dates (93, 101 and 110). Thus, the following inputs are provided for each instant: 69, 80, 93, 101 for 110; 61, 69, 80, 93 for 101, and; 50, 61, 69, 80 for 93).

The dataset is modeled by matrices L and P (see Equation 1), where L contains the last execution moments for q query and P contains the dates to be predicted. In other words, Each element of P presents the date that the query was executed and each row of L presents the last 4 query executions before that date.

$$L = \begin{pmatrix} 50 & 61 & 69 & 80 \\ 61 & 69 & 80 & 93 \\ 69 & 80 & 93 & 101 \end{pmatrix} P = \begin{pmatrix} 93 \\ 101 \\ 110 \end{pmatrix} \tag{1}$$

According to step 1, B datasets are constructed by a resampling scheme on the original data.Taking $B = 4$, the procedure could result on the following datasets.

$$L_1 = \begin{pmatrix} 50 & 61 & 69 & 80 \\ 61 & 69 & 80 & 93 \\ 50 & 61 & 69 & 80 \end{pmatrix} P_1 = \begin{pmatrix} 93 \\ 101 \\ 93 \end{pmatrix} \tag{2}$$

$$L_2 = \begin{pmatrix} 69 & 80 & 93 & 101 \\ 69 & 80 & 93 & 101 \\ 50 & 61 & 69 & 80 \end{pmatrix} P_2 = \begin{pmatrix} 110 \\ 110 \\ 93 \end{pmatrix} \tag{3}$$

$$L_3 = \begin{pmatrix} 61 & 69 & 80 & 93 \\ 61 & 69 & 80 & 93 \\ 69 & 80 & 93 & 101 \end{pmatrix} P_3 = \begin{pmatrix} 101 \\ 101 \\ 110 \end{pmatrix} \tag{4}$$

$$L_4 = \begin{pmatrix} 69 & 80 & 93 & 101 \\ 69 & 80 & 93 & 101 \\ 69 & 80 & 93 & 101 \end{pmatrix} P_4 = \begin{pmatrix} 110 \\ 110 \\ 110 \end{pmatrix} \tag{5}$$

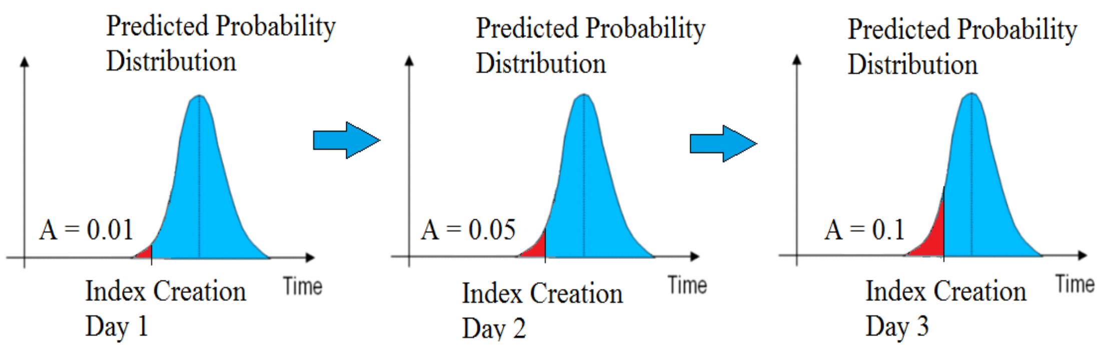
Predicted Probability Distribution
A = 0.01
Index Creation Day 1
Time
Predicted Probability Distribution
A = 0.05
Index Creation Day 2
Time
Predicted Probability Distribution
A = 0.1
Index Creation Day 3
Time

Fig. 2. Decision Support System Example

It is important to observe that the bootstrap is a stochastic procedure and may result in different resample sets. The resample sets previously presented are just illustrative examples.

In step 2, using each dataset, one can obtain 4 different prediction instances. During the execution of step 3, DSIM predicts the next execution of q using all prediction instances and than obtaining 4 predicted dates. Using those dates, a histogram can be built by counting the number of predictions for each date. This is done in step 4. Although, in this example 4 predictions have been used, a real case may employ several instances and several predictions in order to build an informative histogram.

4.5 Step 5: Decision support

During the execution of the decision support step, the predicted histogram provided by the previous step for q is made available to help the decision maker to choose the ideal instant to create efficient indexes to speed up $q's$ execution. Note that creating a index too early increases cost as the index is created with an unnecessary anticipation. On the other hand, planning to create indexes belatedly incurs on the risk of performing the query without the appropriate indexes. In this scenario, the risk acceptance level of the decision maker is an import variable to this problem.

In this work, the risk acceptance level (RAL) is quantified as the probability of executing a query without the corresponding indexes on a given date. The RAL can be calculated as the area under the probability distribution until a given date. Thus RAL is provided as an input parameter. As the decision maker sets the RAL, the system outputs the day corresponding to the RAL configured. Figure 3 provides an illustrative example of this procedure.

In order to compute the most appropriate date to create indexes for a given query, the algorithm depicted in Figure 3 is used. Note that the recommended date for index creation is given by the variable DAY. Taking the predicted probability distribution shown in Figure 3 and a RAL of 0.1 the decision support system calculates the area of the probability distribution before day 1. The result of 0.01 indicates that the risk is below the acceptance level. In this case, the system may move forward and check the next day. On day 2, the probability is 0.05, which is still below the configured RAL. At day 3, the area calculation result in a probability of 0.1. This result is equal to the configured threshold and so the system recommends to create the index on day 3. It is important to notice that in a situation of threshold exceeding at a day n, the system recommends the index creation at day $n-1$.

5. EVALUATION

In order to evaluate the proposed approach we present next a real execution scenario. The idea is to describe and analyze how each step implemented by DSIM works.

Algorithm 2 Decision Support

```
1: DAY ← 0
2: repeat
3:     DAY++
4:     AREA ← Area under the histogram from 0 to DAY
5: until AREA > Risk Acceptance Level (RAL)
6: return DAY
```

Fig. 3. Decision Support Procedure

Table I. The Most Appropriate Index for each Time Consuming Query

Query	Table	Column
Query 1	Warning	warning_type
Query 2	Payment	customer_id
Query 3	Bill	bill_type
Query 4	Customer	fl_active
Query 5	Parcel	parcel_type

5.1 Step 1: Capturing the workload

The workload has been captured from log files of a real database used in a multinational corporation, whose identity may not be revealed for confidentiality reasons. Records of the transactions executed between 2008 and 2010 have been analyzed. The database consists of 1531 tables, occupying 1.5 TB, and it is used by an OLTP application.

5.2 Step 2: Identifying time-consuming queries

We used the same set of time-consuming queries Q and the same set of index structures that were investigated by Medeiros et al. [2012]. This way, Q contains five queries and each query is associated with just one index structure. Each index structure is defined in a certain column of a specific table (see Table I).

5.3 Step 3: Selecting appropriate prediction models for time-consuming queries

In this work we have evaluated three different prediction models: linear regression, multilayer perceptron (MLP) and radial basis function (RBF). For each prediction model we have evaluated just one prediction instance, which was defined using the best configuration (parameter values) showed by Medeiros et al. [2012]. In order to create those instances each date is represented by an integer value that represents a day in the observation period, i.e., a day between the years 2008 and 2010. Thus, the network input and output values range from 1 to 1095 (365 x 3). Besides, we have used a time window of size 4. In other words, the last $k = 4$ query execution days were chosen to compose the input vector for the prediction instances. MLP instances are 1-hidden layer neural network with 10 neuron on the hidden layer. The RBF instances have 5 centroids and a Gaussian kernel.

For each prediction instance created for a query q, we have performed five predictions and the instance with smaller RMSE was linked to q. The model selection for each query q resulted on a linear model for query 1, MLP for queries 2 and 4 and a RBF for queries 3 and 5.

5.4 Step 4: Uncertainty estimation

For each query q we have used the prediction instance selected in the previous step and the boostrap technique. Thereafter, 50 new prediction instances were created and, for each instance, one prediction has been computed. The results of those 50 predictions have been employed to build the prediction histogram for q.

Table II. Confidence Interval and Ratio of Queries Inside Confidence Interval

Confidence Interval	RQI
0.99	1
0.9	0.95
0.8	0.8
0.7	0.75
0.6	0.6
0.5	0.6

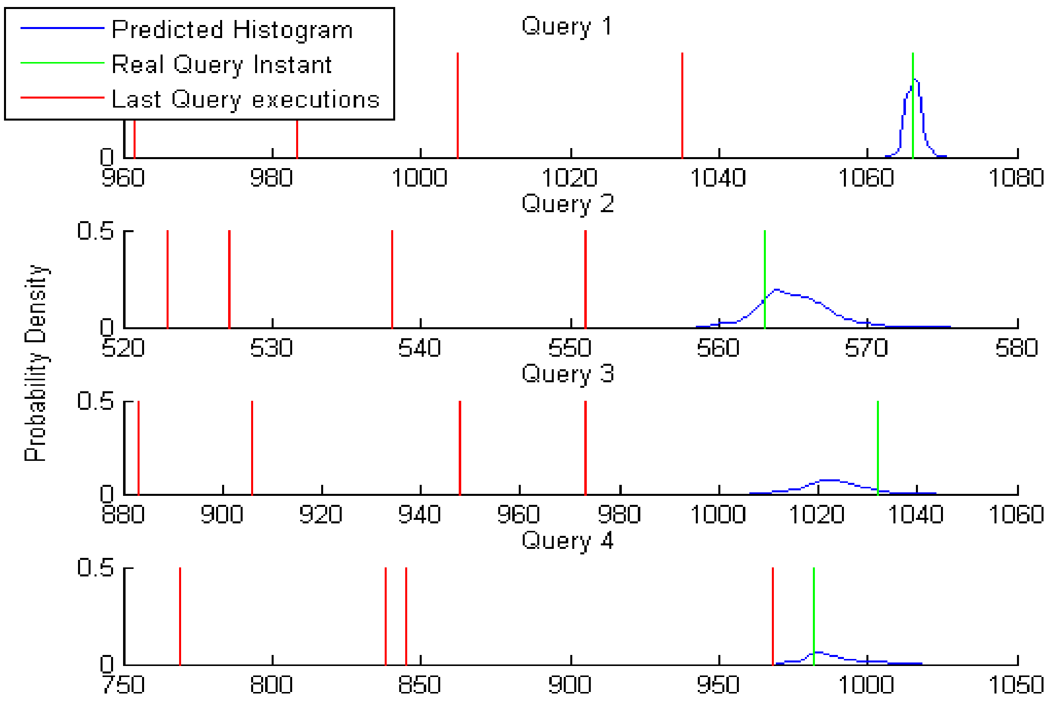

Fig. 4. Sample Predictions for Queries 1, 2, 3 and 4

5.5 Step 5: Decision support

The quantitative evaluation of these predictions can not be done on straightforward way, since the real query execution date is frequently inside the prediction histogram. However, a confidence interval analysis can be done to evaluate how well the uncertainty is fitted by the constructed histograms.

For the confidence interval analysis, a confidence interval c_i is set. For a well designed predicted histogram, it is expected that the rate of real query instants per total of queries is close to c_i. Considering a 99% confidence interval, all 20 real query execution dates fall inside the histograms predicted. This number reduces as the confidence interval is reduced. Table II shows the confidence interval and the ratio of real query execution dates falling inside this interval (RQI).

Observing Table II one can see that the RQI is always higher than the confidence interval. This result indicates that the confidence intervals are not underestimated and can be used as efficient measures of uncertainty.

Figure 4 presents examples of predictions for queries 1 to 4, respectively. The query 5 was not used because H_{q_5} had only 13 executions and this set is short to be used with neural networks. The presented instances were not used during the prediction model learning step.

In Figure 4, one can observe that real query dates are always inside the predicted histogram. Another important fact that should be noticed it that the predicted histogram is sharper for queries whose behavior is closest to periodic. This fact can be seen on query 1. Due to its well behaved occurrences, query 1 could be modeled with a linear model.

For the examples presented in Figure 4 it is possible to demonstrate how the decision support system should work. As a first step, the decision maker should choose a RAL. The RAL determines how much risk the decision maker can take. This risk is defined as the chance that a given query is executed and the index is not created. For instance, taking a RAL of 5%, the system calculates the area under the predicted histograms for a sequence of days. The recommended day is determined when the area surpasses 5%. Thus, DSIM recommends days 1064, 561, 1014 and 975 for queries 1 to 4 respectively. The real dates for each query were 1066, 563, 1032 and 982.

Taking a RAL of 50% incurs in accepting a much higher risk. Although this situation may not be reasonable, it is suitable to illustrate the proposed method. This RAL will lead to the recommended days 1067, 564, 1022 and 985. In this case, we can see that the risk was too high and queries 1, 2 and 4 occurred before an index was created.

It is important to note that it is not possible to compare performance evaluation results between DSIM and the approach proposed by Medeiros et al. [2012]. Although DSIM is based on the work of Medeiros et al. [2012], the main objectives of both approaches are quite different. Medeiros et al. [2012] propose a method to predict the point in time when a given query q will be executed. On the other hand, DSIM delivers a full probability distribution, represented by a prediction histogram, for the day that q will be submitted next time. Based on the prediction histogram and the RAL (specified by the DBA), DSIM indicates the optimal instant to create the necessary indexes for q. Thus, choosing the date to create the index is a DBAs task that could be assisted by DISM results.

6. CONCLUSION AND FUTURE WORK

In this work, we have presented DSIM (Decision Support Mechanism for Index Maintenance) a mechanism to uphold decision-making activities about index maintenance. The main objective of DSIM is to provide a DSS (Decision Support System) to the problem of choosing the best instant to maintain indexes for quasi-periodic time-consuming SQL queries. DSIM provides a risk acceptance configuration feature using a probabilistic framework.

Future works shall explore the possibility of choosing the RAL in an automated way. This choice may be based on the compromise involving the cost of maintaining the index for a given time and the cost of executing a query without having the index. Another direction to be explored addresses the prediction of queries that are not periodic or quasi-periodic. The possible occurrence of light queries before a time-consuming query may be a pattern that could help the prediction. Experiments using association rules and other data mining techniques will de performed.

We want to conclude this article by highlighting that the results of the experiments show the feasibility and effectiveness of DSIM. Besides, the developed mechanism can also be configured to operate in a automatic manner, similar to the approach presented by Medeiros et al. [2012], if the DBA defines the risk acceptance level that will be used.

REFERENCES

Agrawal, S., Chaudhuri, S., Kollar, L., Marathe, A., Narasayya, V., and Syamala, M. Database Tuning Advisor for Microsoft SQL Server 2005. In *Proceedings of the International Conference on Very Large Data Bases*. Toronto, Canada, pp. 1110–1121, 2004.

Alagiannis, I., Dash, D., Schnaitter, K., Ailamaki, A., and Polyzotis, N. An Automated, yet Interactive and Portable DB Designer. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. New York, NY, USA, pp. 1183–1186, 2010.

Bertsimas, D. and Freund, R. *Data, Models, and Decisions: the fundamentals of management science*. Dynamic Ideas, 2004.

Bourouisa, A., Fehama, M., Hossainb, M., and Zhang, L. An Intelligent Mobile Based Decision Support System for Retinal Disease Diagnosis. *Decision Support Systems* 59 (1): 341–350, 2014.

Bruno, N. and Chaudhuri, S. An Online Approach to Physical Design Tuning. In *Proceedings of the IEEE International Conference on Data Engineering*. Los Alamitos, CA, USA, pp. 826–835, 2007a.

Bruno, N. and Chaudhuri, S. Online Autoadmin: (physical design tuning). In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. pp. 1067–1069, 2007b.

Bruno, N. and Chaudhuri, S. Interactive Physical Design Tuning. In *International Conference on Data Engineering*. Long Beach, CA, USA, pp. 1161–1164, 2010.

Chaudhuri, S. and Narasayya, V. Self-Tuning Database Systems: a decade of progress. In *Proceedings of the International Conference on Very Large Data Bases*. pp. 3–14, 2007.

Dageville, B., Das, D., Dias, K., Yagoub, K., Zait, M., and Ziauddin, M. Automatic SQL Tuning in Oracle 10g. In *Proceedings of the International Conference on Very Large Data Bases*. Toronto, Canada, pp. 1098–1109, 2004.

Dahala, K., Almejallia, K., and Hossain, M. A. Decision Support for Coordinated road Traffic Control Actions. *Decision Support Systems* 54 (2): 962–975, 2014.

de Carvalho Costa, R. L., Lifschitz, S., de Noronha, M. F., and Salles, M. A. V. Implementation of an Agent Architecture for Automated Index Tuning. Washington, DC, USA, pp. 1215–1215, 2005.

Drudzdel, M. J. and Flynn, R. R. Decision Support Systems. Encyclopedia of Library and Information Science, 2nd edition, Allen Kent (ed), New York: Marcel Dekker Inc., 2002.

Efron, B. Bootstrap Methods: another look at the jackknife. *The Annals of Statistics* 7 (1): 1–26, 1979.

Kima, D. J., Ferrinb, D. L., and Rao, H. R. A Trust-based Consumer Decision-making Model in Electronic Commerce: the role of trust, perceived risk, and their antecedents. *Decision Support Systems* 44 (2): 544–564, 2014.

Luhring, M., Sattler, K.-U., Schmidt, K., and Schallehn, E. Autonomous Management of Soft Indexes. In *Proceedings of the IEEE International Conference on Data Engineering*. Washington, DC, USA, pp. 450–458, 2007.

Maier, C., Dash, D., Alagiannis, I., Ailamaki, A., and Heinis, T. Parinda: an interactive physical designer for postgresql. In *International Conference on Extending Database Technology*. New York, NY, USA, pp. 701–704, 2010.

Medeiros, A., Saraiva, A., Campos, G., Holanda, P., Monteiro, J. M., Brayner, A., and Lifschitz, S. Proactive Index Maintenance: Using Prediction Models for Improving Index Maintenance in Databases. *jidm* 3 (3): 255–270, 2012.

Morelli, E. M. T., Monteiro, J. M., Almeida, A. C., and Lifschitz, S. Automatic Reindexation in Relational DBMSS. In *Proceedings of the Brazilian Symposium on Databases*. pp. 31–45, 2009.

Salles, M. A. V. and Lifschitz, S. Autonomic Index Management. Seattle, WA, USA, pp. 304–305, 2005.

Sattler, K.-U., Geist, I., and Schallehn, E. Quiet: continuous query-driven index tuning. In *Proceedings of the International Conference on Very Large Data Bases*. pp. 1129–1132, 2003.

Sattler, K.-U., Schallehn, E., and Geist, I. Autonomous Query-Driven Index Tuning. Washington, DC, USA, pp. 439–448, 2004.

Schnaitter, K., Abiteboul, S., Milo, T., and Polyzotis, N. Colt: Continuous on-line tuning. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. Chicago, USA, pp. 793–795, 2006.

Schnaitter, K., Abiteboul, S., Milo, T., and Polyzotis, N. On-Line Index Selection for Shifting Workloads. Istanbul, Turkey, pp. 459–468, 2007.

Valentin, G., Zuliani, M., Zilio, D. C., Lohman, G. M., and Skelley, A. DB2 Advisor: an optimizer smart enough to recommend its own indexes. In *Proceedings of the IEEE International Conference on Data Engineering*. San Diego, CA, USA, pp. 101–110, 2000.