An Automated, yet Interactive and Portable DB designer

Ioannis Alagiannis[1]
ioannis.alagiannis@epfl.ch

Debabrata Dash[1, 2]
ddash@cmu.edu

Karl Schnaitter[3]
karlsch@soe.ucsc.edu

Anastasia Ailamaki[1]
natassa@epfl.ch

Neoklis Polyzotis[3]
alkis@ucsc.edu

[1] École Polytechnique Fédérale de Lausanne

[2] Carnegie Mellon University

[3] UC Santa Cruz

ABSTRACT

Tuning tools attempt to configure a database to achieve optimal performance for a given workload. Selecting an optimal set of physical structures is computationally hard since it involves searching a vast space of possible configurations. Commercial DBMSs offer tools that can address this problem. The usefulness of such tools, however, is limited by their dependence on greedy heuristics, the need for a-priori (offline) knowledge of the workload, and lack of an optimal materialization schedule to get the best out of suggested design features. Moreover, the open source DBMSs do not provide any automated tuning tools.

This demonstration introduces a comprehensive physical designer for the PostgreSQL open source DBMS. The tool suggests design features for both offline and online workloads. It provides close to optimal suggestions for indexes for a given workload by modeling the problem as a combinatorial optimization problem and solving it by sophisticated and mature solvers. It also determines the interaction between indexes to suggest an effective materialization strategy for the selected indexes. The tool is interactive as it allows the database administrator (DBA) to suggest a set of candidate features and shows their benefits and interactions visually. For the demonstration we use large real-world scientific datasets and query workloads.

Categories and Subject Descriptors

H.2.2 [**Physical Design**]: Access Methods

General Terms

Algorithms, Performance, Design

Keywords

Physical Design Tuning, Continuous Tuning, Index Interaction

1. INTRODUCTION

Database Management Systems (DBMSs) have been widely deployed the last years and the more complicated the database applications become the more important the physical design is. Selecting indexes, materialized views, horizontal and vertical partitions that can enhance performance in a workload is a challenging optimization problem especially if storage resources are limited. Manual physical design is both time consuming and very tedious, as the DBA needs to find the benefits of individual design features. Thus, the need for automating physical design tools has become more demanding than ever.

There are several commercial tools that offer automating tuning with several features [1][3][10]. These tools are based on greedy heuristics. They allow what-if design exploration and have useful user interface. Although these greedy heuristics make the existing design tools practical, they prune away large fractions of the search space and often suggest locally optimal solutions instead of the globally optimal one. On the other hand, little work has been done in providing similar tools for open source DBMSs, such as PostgreSQL and MySQL. Thus, one has to face the dilemma of selecting an expensive commercial DBMS that provides automating tools or an open source DBMS whose lack of automated tools might increase the operational cost in the long run. In addition, in the real world, the queries running on a database evolve over time. Thus, the suggested physical designs may become obsolete and require re-optimizations. Since the re-optimization is expensive, it is desirable to have a lightweight online tuning tool to monitor the query evolution and frequently to optimize the design. Finally, since the design features, such as indexes, typically take considerable amount of time to build, it is essential to schedule their construction in a way that maximizes their benefits.

We address the above requirements by developing an automated and interactive physical design tool for the PostgreSQL open source DBMS. Our tool incorporates miscellaneous algorithms and techniques to improve overall performance and provided features. Given a database, a set of queries and resource constraints, our tool suggests a near optimal configuration. It uses CoPhy [4] to suggest the indexes. CoPhy develops a convex combinatorial optimization formulation for the problem of suggesting indexes and then employs mature existing techniques to solve it. CoPhy allows to trade off execution time against the quality of the suggested solutions. The tool uses AutoPart [8] to suggest optimal partitions for the workload. It also incorporates a lightweight online physical designer—COLT [11]—to monitor the performance of the queries and suggest changes to the physical design when the existing design is suboptimal for the workload. It also suggests an effective materialization schedule for the suggested indexes using their interactions [12]. Finally, it makes the design selection process more interactive by allowing the DBA or a novice user to specify a candidate set of indexes and study the benefit of these indexes efficiently by using simulated what-if indexes, and visualizing their interactions. The user can

SIGMOD'10, June 6–10, 2010, Indianapolis, Indiana, USA.

also control the physical design search by suggesting a candidate set of indexes as the starting point of the search algorithm in the designer.

In this demonstration, we use PostgreSQL because of its popularity and its relatively mature cost-based optimizer. However, the tool is designed so that it can be ported to any relational DBMS, which offers a query optimizer, a way to extract and create statistics, and control over join operations. To run the aforementioned diverse techniques on PostgreSQL, we modify the query optimizer to add what-if capabilities. The what-if capabilities simulate the original design features without actually building them, hence enabling efficient exploration of the feature space. We then extend the INUM [9] cache-based cost model to cache table partitions and partial plans to further increase the efficiency of the selection tool by orders of magnitude.

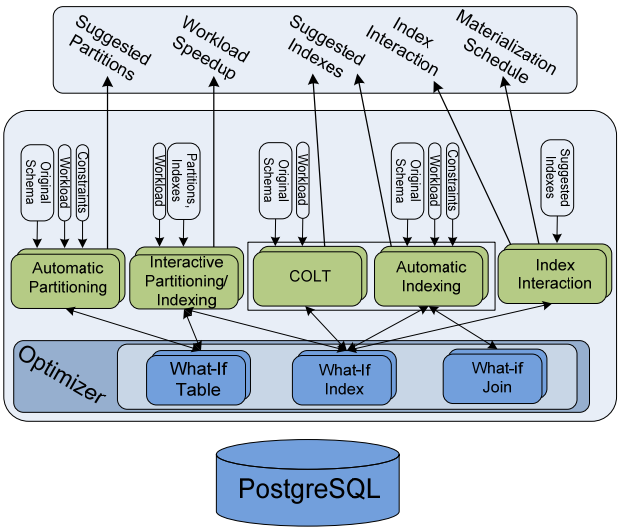

Figure 1. System Architecture

Demonstration Structure: This demonstration presents a new tool which extends PostgreSQL by adding automatic physical design features. Because scientific datasets are usually very big and involve complex queries, we demonstrate the effectiveness of the tool using a real-world SDSS [13] dataset and query workload. We demonstrate three physical design scenarios. In the first scenario, the DBA manually selects the combination of design features and the tool determines the benefit of using that combination. The second one finds the optimal indexes and partitions for a given query workload. It also suggests a schedule to implement the suggested indexes. The last one continuously monitors the performance of the DBMS under incoming queries, and suggests new indexes when the indexes offer sufficient workload speedup.

The rest of the paper is organized as follows: Section 2 presents the related work; Section 3 describes the system architecture; Section 4 presents the demonstration scenarios; and Section 5 concludes the paper.

2. RELATED WORK

Researchers propose several techniques for automated physical design. Due to space limit, we list only the recent commercial automated physical design tools, such as the Data Tuning Advisor (DTA) for SQL Server [1], the Design Advisor for DB2 [14], and the SQL Access Advisor for Oracle[10]. These commercial tools use what-if design features [5], scale by pruning away the space in greedy manner, and do not support online physical design.

The automated physical designers for open source DBMSs are relatively new compared to commercial ones. Monterio et al. implement and design an index suggestion tool for PostgreSQL [7]. They, however, assume the size of the indexes to be zero, which severely affects the accuracy of the optimizer when what-if indexes are used. Kao et al. propose changing the optimizer to store the access paths and suggest the frequently requested access paths [6]. This, however, requires drastic changes to the optimizer, and does not explore new access paths.

Similar to COLT, Bruno et al. suggest online index tuning [2]. They, however, use proprietary interfaces with SQL Server, which are not portable to PostgreSQL and they also provide a heuristic method for index interaction, but Schnaitter et al. [12] extends the concept by comprehensively analyzing its properties and providing algorithms to schedule the suggested indexes.

3. SYSTEM ARCHITECTURE

In this section we outline the basic components of our system, their role and how they interact with each other. A high-level architecture of our system is illustrated in Figure 1.

3.1 What-if Component

The what-if component is a basic component of our architecture and all the other components are attached to it. It allows for simulating the potential benefit from the presence of physical structures such as partitions and indexes without having to construct them. To achieve this, we modify PostgreSQL query optimizer and evaluate the cost of queries using what-if analysis. The optimizer computes the execution plan of a query, assuming that what-if indexes and tables are implemented in the database. Thus, we escape the cost of explicitly building a structure. This component consists of three sub-components: a) the what-if index component which is used for index simulation, b) the what if table component which simulates the presence of vertical and horizontal partitions and c) the what-if join component which controls the join methods in the query execution plan.

3.2 Index Recommendations

Our tool provides two ways of index tuning: the automatic index suggestion component and the continuous tuning component for offline and online index recommendations respectively.

3.2.1 Automatic Index Suggestion

The automatic index suggestion component uses Cophy [4]. It takes as input the query workload, the physical design and size constraints. Then, it develops a convex combinatorial optimization formulation for the index selection problem. While the candidate indexes are analyzed, a cache-based cost model (INUM) speeds up the cost estimation process by caching and reusing intermediate results. The component returns a set of suggested indexes.

3.2.2 Continuous Tuning

The continuous tuning component uses COLT [11]. COLT is an online index selection framework that continuously monitors incoming streams of queries, evaluates the benefit from adopting different indexes and proposes the most promising configuration

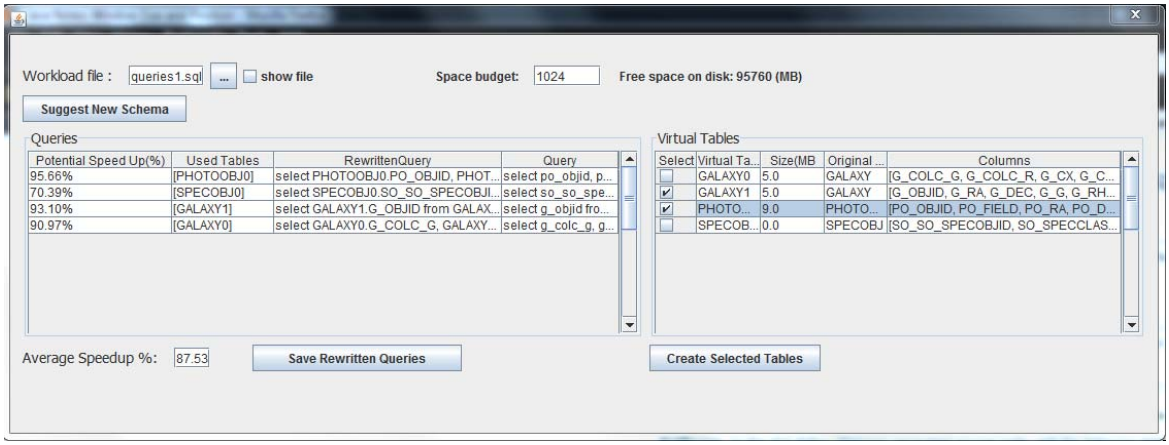

Figure 3. Automatic Partition Suggestion Interface

of single-column indexes respecting the workload and potential space constraints. Continuous tuning component operates additionally to the rest of our tool and it can be enabled or disabled in accordance with workload or administrator's will. Initially, it examines workload traffic in a preconfigured database system. If it detects a change in indexes that can improve performance, an alert message is sent. The new proposed configuration includes only single column indexes. So, whether this configuration would be adopted or not, depends on the DBA. She might have to choose between the new single column index configuration proposed by the continuous tuning component and a multicolumn index configuration initially proposed by the automatic index suggestion component.

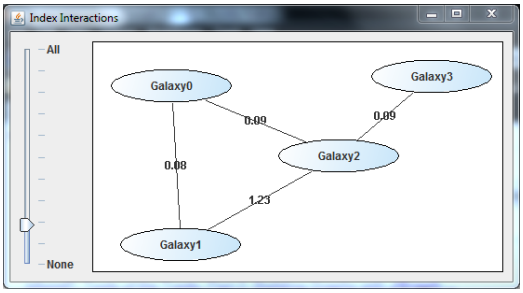

Figure 2. Index Interaction

3.3 Automatic Partition Suggestion

The automatic partition component uses the AutoPart technique [8]. This component receives as input the query workload, the original physical design, and constraints such as space limitations for replicating columns in the partition. It produces vertical and horizontal partitions which can optimally improve the execution time for the aforementioned workload. In this component, we have also extended the INUM cost model to include partitions.

3.4 Interactive Partitioning/Indexing

The interactive partitioning/indexing component receives as input the original schema and a query workload and enables the user to select indexes and partitions interactively. Apart from that, the average workload benefit and the individual queries benefits from the new schema are computed by using the indexes' and partitions' cost model in a unified approach.

3.5 Index Interaction

The index interaction component embeds to our system the functionality of two database tuning tools introduced by Schnaitter et al. [12]. Both tools exploit knowledge about index interaction and are combined with the automatic index suggestion and interactive partitioning/indexing component. The first tool receives as input the recommended indexes from one of the aforementioned components and provides DBA with visualization of interactions between them. The goal is to help the DBA gain some understanding about the interactions between the specified indexes. The second tool schedules the materialization of suggested indexes. The rationale that lies behind the use of this tool is that an appropriately scheduled materialization of indexes can lead to higher benefit in contrast with a schedule that does not take into account index interaction.

4. DEMONSTRATION

In this section, we describe three application scenarios that demonstrate how the features of our tool are used to tune the physical design of a database.

Scenario #1: The goal of this scenario is to estimate the potential benefits of a new physical design. The user provides the query workload and the original physical schema. Then, she creates several what-if partitions and indexes using the tool's interface. Now, the tool presents the benefits from using the new physical design for the particular workload. The user can examine interactions between the what-if indexes as visualized by the Index Interaction component and save the rewritten queries for the new table partitions. Figure 2 presents how index interaction is visualized. We use an undirected graph in which the vertices of the graph represent indexes and the weights of the edges are the degree of interaction for a pair of indexes. If the graph has too many edges, the user can dynamically change the number of interactions that are being displayed.

Scenario #2: In this scenario, the user provides the query workload, the original physical schema and size constraints. Then, the tool recommends a set of indexes and partitions which maximize the performance. The interface presents the list of suggested indexes and partitions, the average workload benefit and the benefit per query. Again, the interaction between the proposed indexes is illustrated. The user has the option to

physically create the suggested partitions and indexes. In the case of indexes, a materialization schedule becomes available. This schedule takes into consideration the index interactions to find a beneficial order of index materialization. Figure 3 shows an example of how suggested partitions are presented to the user. The list of suggested partitions is displayed in the right panel of the user interface. The user can examine the individual query benefit and the average workload benefit in case she adopts the suggested changes to the schema. Additionally, the user has the option of physically creating the suggested partitions and save the workload queries according to the new partitions.

Scenario #3: This scenario focuses on the use of the continuous tuning component. This component monitors the behavior of the system when the workload changes and suggests changes to the set of indexes. Our tool presents the change in system's performance accruing from adopting the new suggested indexes.

5. CONCLUSION

In this demonstration, we presented a new automating physical design tool for open source DBMSs. The tool uses what-if analysis to simulate potential changes to the schema and integrates algorithms for suggesting indexes and partitions. It can modify the schema if there is a change in the workload, visualize interaction between indexes and present a materialization schedule for indexes. We demonstrate the tool on three different scenarios using scientific datasets and present the functionality using the tool's interface.

6. ACKNOWLEDGMENTS

This work was partially supported by Sloan research fellowship, NSF grants CCR-0205544, IIS-0133686, and IIS-0713409, an ESF EurYI award, and SNF funds.

7. REFERENCES

[1] S. Agrawal et al. Database Tuning Advisor for Microsoft SQL Server 2005. In Proceedings of the International Conference on Very Large Databases (VLDB), 2004.

[2] N. Bruno and S. Chaudhuri. An Online Approach to Physical Design Tuning. ICDE'07.

[3] N. Bruno and Surajit Chaudhuri. Automatic physical database tuning: a relaxation-based approach. In Proceedings of the SIGMOD Conference, 2005.

[4] D. Dash, A. Ailamaki. CoPhy: Automated Physical Design with Quality Guarantees. Technical Report CMU-CS-10-109.

[5] S. Finkelstein,M. Schkolnick,P. Tiberio: Physical database design for relational databases. ACM ToDS. 1988.

[6] Kao, K., Liao, I. 2009. An index selection method without repeated optimizer estimations. Inf. Sci. 179, 13 (Jun. 09)

[7] Monteiro, J. M., Lifschitz, S. and Brayner, A.: An Architecture for Automated Index Tuning. In V Ph.D. and M.S. SBBD, 2006.

[8] S. Papadomanolakis, A. Ailamaki, AutoPart: Automating Schema Design for Large Scientific Databases Using Data Partitioning, 6th International Conference on Scientific and Statistical Database Management (SSDBM'04), 2004.

[9] S. Papadomanolakis, D. Dash, A. Ailamaki. Efficient Use of the Query Optimizer for Automated Physical Design. VLDB 2007.

[10] Performance Tuning using the SQLAccess Advisor. http://www.oracle.com/technology/products/bi/db/10g/pdf/twp_general_perf_tuning_using_sqlaccess_advisor_10gr1_1203.pdf

[11] K. Schnaitter, S. Abiteboul, T. Milo, and N. Polyzotis. Colt: continuous on-line tuning. In proceedings of the 2006 ACM SIGMOD, pages 793–795, 2006.

[12] K. Schnaitter, N. Polyzotis, L. Getoor: Index Interactions in Physical Design Tuning: Modeling, Analysis, and Applications. PVLDB 2(1): 1234-1245 (2009).

[13] Sloan Digital Sky Survey, http://www.sdss.org/

[14] D. C. Zilio, J. Rao, et al. DB2 Design Advisor: Integrated Automatic Physical Data-base Design. VLDB'04.