PARINDA: An Interactive Physical Designer for PostgreSQL

Cristina Maier Debabrata Dash Ioannis Alagiannis Anastasia Ailamaki Thomas Heinis
Ecole Polytechnique Fédérale de Lausanne, Switzerland
{cristina.maier,debabrata.dash,Ioannis.alagiannis,anastasia.ailamaki,thomas.heinis}@epfl.ch

ABSTRACT

One of the most challenging tasks for the database administrator is to physically design the database to attain optimal performance for a given workload. Physical design is hard because it requires the selection of an optimal set of design features from a vast search space. There have been many commercial tools available to automatically suggest the physical design, for a given a set of queries. These tools are, however, based on greedy heuristic pruning, which reduces their usefulness. Furthermore, they are not interactive, as the APIs to simulate the indexes and tables are product specific and hidden from the database administrators. Finally, all these tools are built specifically for commercial systems and there is lack of automated physical designers for open source DBMSs. In this demonstration we introduce –PARINDA - an interactive physical designer for an open source DBMS. Given a workload containing a set of queries, this tool allows the DBA to efficiently simulate various physical design features and get immediate feedback on their effectiveness. It also incorporates recent advances in non-greedy physical design techniques to provide close to optimal suggestions. Although it has been prototyped for several different DBMSs, we demonstrate the usefulness and efficiency of the tool while running on the open source DBMS—PostgreSQL--using large real-world scientific datasets and query workloads.

1. INTRODUCTION

Physical design of databases seeks to optimize the performance of the database by adding design features, such as horizontal and vertical partitions, indexes, or materialized views, in order to speed up the queries in the workload. Without support from the DBMS, the only way a database administrator (DBA) can decide on the optimal physical design structures is to build them manually, and then estimate the query execution time for combinations of the design features. This task is both cumbersome and expensive, as building design features, such as indexes takes a considerable amount of time and planning. Therefore, automating the physical design selection is crucial.

Researchers have proposed several automated physical design techniques for commercial DBMSs [8][11][12]. They all scale using greedy heuristics to prune away the search space. The greedy pruning makes the tools feasible, but reduces their usefulness by pruning away many useful candidates. They also do not allow the DBA to experiment with the design features without actually building them. Finally, there has been no such designer tool for an open source DBMS. Even though the cost of the DBMS is a major factor in deciding for an open source DBMS, the lack of such automated tools makes the operation of an open source DBMS more expensive than a commercial system.

EDBT 2010, March 22-26, 2010, Lausanne, Switzerland.

In this demonstration, we introduce a new automated physical design tool – PARINDA (PARtition and INDex Advisor) - for an open source DBMS. Given a database and a set of queries, the tool does not prune away the candidate space greedily. Hence, it searches through all useful candidate features before suggesting the optimal set of features. It allows the DBA to interactively estimate the benefit of new physical design features by simulating the design features efficiently. Finally, it automatically rewrites the queries to get the full benefit of the suggested design features.

In this demonstration, we use PostgreSQL as the underlying DBMS for PARINDA. We do so because compared to other open source DBMS, PostgreSQL has a mature cost-based optimizer.

PARINDA first modifies the optimizer to enable what-if physical design features. These features are not actually built on the disk. They are simulated by creating statistics in the DBMS catalog. Since the query optimizer primarily deals with statistics, it cannot differentiate between the real design features and the what-if ones. Therefore, these what-if structures allow the DBA to estimate the benefit they would get if the structures were actually present in the database. Simulating the structures makes the operations orders of magnitude faster and allows the DBA to explore a larger solution space interactively.

Even with the what-if design features, the search space is too large for the DBA to manually find the optimal set of features. Solving the automated physical design problem is computationally hard [9] as well. We implement two practical state-of-the-art search techniques to search for the optimal set of features, which use efficient heuristics to search for close to optimal design features. To search for the optimal set of partitions, we use the AutoPart technique [7] and to find the optimal set of indexes we use the ILP technique [10]. Using these techniques on analytical queries, we achieve speedups ranging from 2x to 10x.

Demonstration Structure: This demonstration presents a new tool which extends PostgreSQL by adding automatic physical design features. Because scientific data sets are usually very big and involve complex queries, we demonstrate the effectiveness of the tool using a real-world SDSS [1] 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 the combination. The second one finds the optimal partitions for a

given query workload. And the last one automatically finds the optimal indexes for the workload.

Organization: The rest of the paper is organized as follows: Section 2 discusses related work. We describe the overall system architecture and the interaction of various components in Section 3. Section 4 discusses the demonstration scenarios in detail and we conclude in Section 5.

2. RELATED WORK

Researchers have proposed many techniques for automated physical design for the last three decades. Due to space restriction, we list only the recent commercial automated physical design tools, such as Data Tuning Advisor (DTA) for SQL Server [8], Design Advisor for DB2 [11], and SQL Access Advisor for Oracle [12]. All these tools use what-if design features, which were first proposed by Finkelstein et al. [2]. Design Advisor also provides a set of candidate design features, given a set of queries. SQL Access Advisor also contains a SQL tuning technique, which changes the SQL to make it perform better on the database. All these commercial tools are based on greedy heuristics, and do not allow the DBA to directly simulate the design features.

The automated physical designers for open source DBMSs are relatively new compared the commercial ones. Recently Thiem et al. proposed an automated physical designer for the Ingres DBMS [5]. Their focus is more on integration of performance monitoring and tuning, instead of pure physical design. Also, their tool does not suggest partitions. Monterio et al. implement and design an index suggestion tool for PostgreSQL [3]. They, however, do not compute the size of the indexes accurately, and assume it to be zero. This severely affects the accuracy of the optimizer using their what-if indexes. Kao et al. propose changing the optimizer to store the optimizer access path decisions in a data structure and suggest the frequently requested access path [4]. This, however, requires drastic changes to the optimizer, and cannot suggest indexes which are not applicable to the existing access paths. COLT [13] also suggests indexes on PostgreSQL, but limits itself to only single column indexes whereas PARINDA can suggest multicolumn indexes.

3. SYSTEM ARCHITECTURE

This section describes the system architecture at a very high level and then discusses the important components of the system in more detail.

Figure 1 shows the architecture of the PARINDA tool. We modify the PostgreSQL query optimizer to add the what-if components.

The what-if components are used to simulate physical partitions, indexes, and presence or lack of join methods. There are three components using the what-if components: the automatic indexing component, the automatic partitioning component and the interactive partitioning/indexing component.

The automatic partitioning component takes as input the query workload, the original physical design, and several DBA defined constraints such as the maximum space taken by replicated columns in the partitions. The output is composed of the new partitions which optimally improve the workload execution time, and the new rewritten queries reflecting the new partitions.

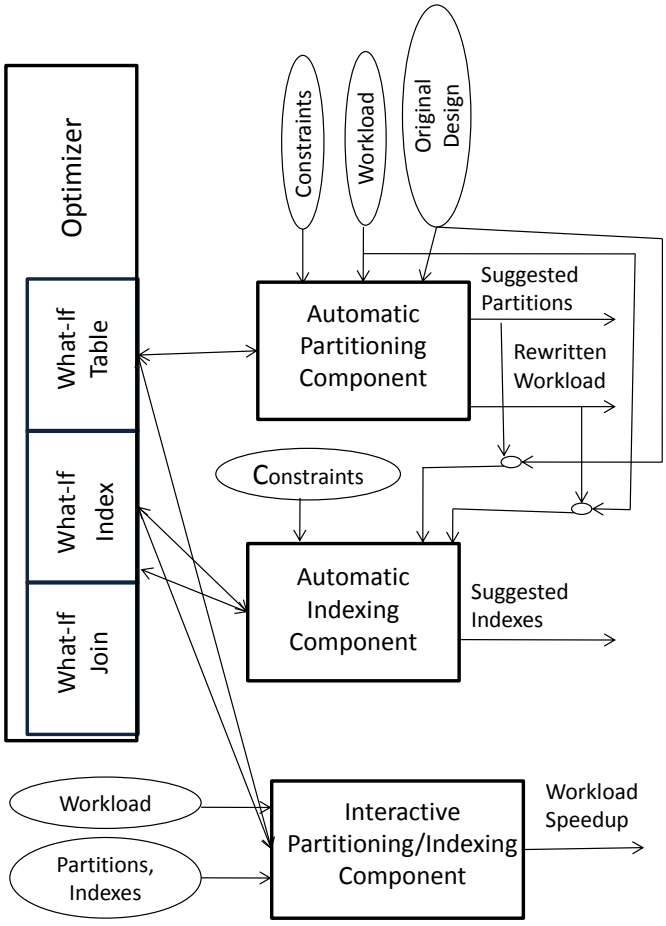

Figure 1 System Architecture for PARINDA

The automatic index component has as input the query workload, the physical design and a size constraints. The output represents the set of suggested indexes.

The input to the interactive partitioning/indexing component is given as the query workload and the original design. It produces a new design and also estimates the benefit of using the new design.

We now describe in the detail the components in the architecture. We skip the interactive component, since its functionality involves only invoking the what-if features and measuring the benefits.

3.1 The PostgreSQL Query Optimizer

The PostgreSQL quey optimizer is the component responsible for generating the execution plan for a SQL query. The optimizer chooses the plan based on the statistics of the original tables and indexes.

In the optimization process, the optimizer first analyzes the query and rewrites it if possible, then builds a structure for storing the statistical information for all the physical design features available for a table. Subsequently it makes decisions to use those design features based on the query structure and statistics. Before making the decisions, the optimizer allows the developer to override the information about physical design by using several function

'hooks'[1]. The hooks can be replaced at runtime with functions that insert new stastistics information into the list of physical design features. This makes the optimizer believe that the newly inserted data regarding the what-if indexes and what-if tables are present in the database. Then, the optimizer selects the execution plans using the statistics from the what-if features.

3.2 What-If Design Features

As Figure 1 shows, the what-if features are divided into three main components, which we descibe next.

What-If Index Component: This component is used for index simulation. The component expects the what-if index definitions along with the query on which the indexes are used as input. Then it computes the number of pages for the indexes using the following formula:

$$Pages = \frac{(o + \sum_{c \in I} (size(c) + align(c))) \times R}{B} \quad (1)$$

Where *o* is the overhead of each row in the index including the rowid pointer back to the main table, *c* is a column in the index *I*, the function *size* finds the average size of the column c in the table, and the function *align* adds extra space to align the values in the disk. The alignment depends on the columns appearing before the current column in the index. *R* is the number of rows in the table, and *B* is the page size. In PostgreSQL 8.3, *o* is 24 and default value of *B* is 8192. We compute only the sizes of the leaf pages, and ignore the internal pages of the B-Tree index, since they affect the relative page sizes only on very small indexes. The optimizer computes histogram statistics about the columns from the statistics of the base table, therefore, we do not compute them.

What-If Table Component: This component is used for partition simulations. Since PostgreSQL does not allow partitions in the table, we simulate the paritions by simulating new tables. These tables contain the primary keys of the original table, so that the full table can be reconstructed from the partitions. The statistics of the original table are used to compute the statistics for the new partitioned table. The number of pages is approximated by using a formula simular to Equation 1. Unlike the what-if indexes, which are completely constructed inside the optimizer, we build empty what-if tables so that the query parser recognizes the new tables and parses the SQL input. At the optimization time we insert the statistics about the new table, making the planner 'believe' the table really exists with data on disk.

What-If Join Component. This is used to control the join methods to be used in the execution plan of the query. This is needed for the INUM (Section 3.4) algorithm from the Automatic Index Suggestion component. INUM caches two plans for each scenario—one with nested-loop enabled and one with nested-loop disabled. We enable and disable the nested-loop join method using the flags offered by the optimizer.

3.3 Automatic Partition Suggestion

The automatic partition component uses the AutoPart technique proposed by Papadomanolakis et al. [7]. This technique partitions the tables in such a way that the workload exeution time improves optimally. First, the component determines the atomic fragments. Atomic fragments are the 'thinnest' possible fragments of the partitioned tables, and they are accessed atomically. This is the first version of the selected fragments. Then, the algorithm improves the initial selected fragments with composite fragments. In the fragments generation step a set of composite fragments are determined. The composite fragments are created by combining atomic fragments with fragments selected in a previous iteration or by combining atomic fragments with atomic fragments. An automatic query rewriter is used to rewrite the original workload for the composite fragments. In the fragment selection step the composite fragments are evaluated using the what-if structures. The replication constraints are considered. The fragments which give the highest improvement for the workload are chosen. Then, the algorithm iterates through the fragments generation and fragments selection steps. Each time, the fragments chosen in the previous step are expanded. The algorithm stops when no more improvement is found. The optimal table partitions are suggested to the user.

3.4 Automatic Index Sugestion

The automatic index suggestion component uses the ILP technique proposed by Papadomanolakis et al. [10]. In this technique the index selection problem is mapped to an integer-linear optimization program, and solved using standard combinatorial solvers. First, the component determines a large set of candidate indexes by analyzing the workload. It then computes the benefit of using a subset of those indexes for different queries. Since this process requires millions of query cost estimations, ILP uses a cache-based cost model (INUM [6]) to speed up the cost estimation process. Using INUM, ILP estimates the costs of millions of physical designs in the order of minutes instead of days. Once the benefits are computed, it constructs an integer-linear program (ILP). The ILP contains the accuracy constraints for the indexes, such that only the one access path is selected for each table in a query, and other user-supplied constraints, such as constraints on the total size of the design features, and their update costs. The program is then solved by a standard off-the-self combinatorial optimization solver and the optimal set of indexes are suggested to the user. Typically ILP outperforms the greedy algorithms on workloads containing a large number of queries. This efficiency is a direct result of INUM's cache-based cost model.

4. DEMONSTRATION

This section describes the demo set up and the scenarios. We use a 5% sample of the SDSS DR4[2] dataset with about 150GB of data in it. For the query workload we use a set of 30 prototypical queries. The database runs on PostgreSQL 8.3 running on a Windows platform. This demonstration presents three possible scenarios.

Interactive Partition/Index Selection Scenario. This scenario estimates the benefit of a new physical design feature. In Figure 2 we present the GUI of this scenario. The user inputs the query workload file and the original physical design. Then, she creates several what-if table partitions and several what-if indexes on the original physical design. The workload is evaluated for the new physical design. The average workload benefit and the individual queries benefits are displayed. The user can save the rewritten queries for the new table partitions. She also has the option to

[1] http://www.postgresql.org/docs/8.1/interactive/index.html

[2] http://www.sdss.org/dr4/

compare the execution plan of the what-if design with the execution plan of the same materialized physical design.

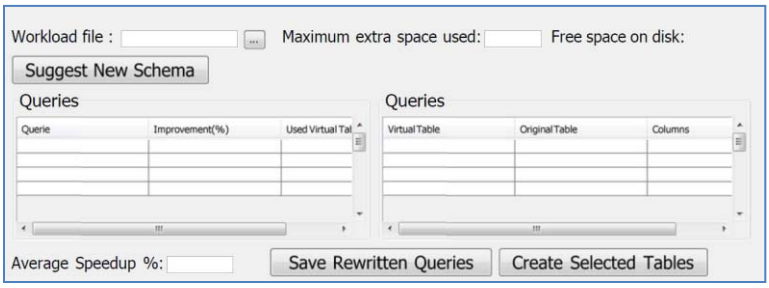

Figure 2 Automatic partition suggestion interface

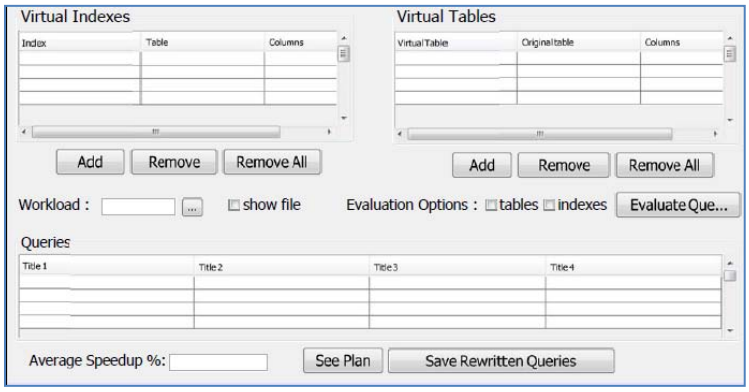

Figure 3 Interactive index and partition selection

This way the accuracy of the physical design simulation is verified. This scenario allows the DBA to manually test small set of candidates to use certain domain knowledge, or to slightly modify the automatic suggestions.

Automatic Partition Suggestion Scenario. This scenario suggests the table partitions which improve the workload queries' execution time optimally. In Figure 3 we present the scenario GUI. The user inputs a workload file, an original physical design and a size constraint. Note that the user does not provide the partition information on this screen. The output consists of the suggested table partitions (using techniques described in Section 3.3), the average workload benefit, and the individual query benefit. For each query, the lists of the suggested partitions used are displayed. The user has the option to physically create on disk the suggested partitions and to save on disk the rewritten workload queries for the new partitions.

Automatic Index Suggestion Scenario. In this scenario a set of indexes which improve the workload queries' execution time optimally is suggested. The GUI of this scenario is very similar to the GUI in the Figure 3, except that it suggests indexes instead of partitions. The inputs are the workload file, a size constraint, the original physical design, and total extra space that the generated indexes can occupy on the disk. The component displays the suggested set of indexes (using techniques described in Section 3.4), the average workload benefit, and the individual query benefit. For each query the list of the used suggested indexes is mentioned. The user has the option to physically create the suggested set of indexes on disk.

5. CONCLUSION

This demonstration introduces a new physical design tool for an open source DBMS. First, the tool implements what-if design features on the DBMS by simulating the statistics of those features and allowing the DBA to check the effectiveness of the design features in an efficient manner. Then it integrates the automatic partitioning mechanism of AutoPart tool to suggest the partitions for a given query set. It also rewrites the input queries to match with the suggested partitions. Finally, it suggests indexes by building an integer-linear program and solves it using a standard off-the-shelf combinatorial solver. We demonstrate the effectiveness of the tool on three different scenarios matching the three functionalities of the tool on a real-world scientific dataset and query workload.

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.*

6. REFERENCES

[1] http://www.sdss.org

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

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

[4] Kao, K. and Liao, I. 2009. An index selection method without repeated optimizer estimations. *Inf. Sci* 2009

[5] Thiem, A. and Sattler, K. An Integrated Approach of Performance Monitoring for Autonomous Tuning. ICDE 2009.

[6] Stratos Papadomanolakis, Debabrata Dash, Anastasia Ailamaki, "Efficient Use of the Query Optimizer for Automated Physical Design", VLDB 2007.

[7] Stratos Papadomanolakis, Anastassia Ailamaki, "AutoPart: Automating Schema Design for Large Scientific Databases Using Data Partitioning", SSDBM 2004

[8] Nicolas Bruno and Surajit Chaudhuri. Automatic physical database tuning: a relaxation-based approach. SIGMOD 2005.

[9] Surajit Chaudhuri, Mayur Datar, and Vivek Narasayya. Index selection for databases: A hardness study and a principled heuristic solution. IEEE TKDE, 2004.

[10] Stratos Papadomanolakis and Anastassia Ailamaki. An Integer Linear Programming Approach to Database Design. SMDB'07.

[11] Daniel C. Zilio, Jun Rao, Sam Lightstone, Guy M. Lohman, Adam J. Storm, Christian Garcia-Arellano and Scott Fadden. DB2 Design Advisor: Integrated Automatic Physical Database Design. VLDB'04.

[12] 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

[13] Schnaitter, K., Abiteboul, S., Milo, T., Polyzotis, N.: COLT: continuous on-line tuning. In Proceedings of ACM SIGMOD Conference 2006.