A tool for automatic index selection in Database Management Systems

Wendel Goes Pedrozo
Department of Informatics
Federal University of Technology
Apucarana, Paraná, Brazil
wpedrozo@utfpr.edu.br

Maria Salete Marcon Gomes Vaz
Department of Informatics
State University of Ponta Grossa
Ponta Grossa, Paraná, Brazil
salete@uepg.br

***Abstract*—A task commonly undertaken by database administrators to speed up the performance of commands submitted to a DBMS is the selection of indexes for the tables. This paper introduces the concept of how to use the Database Management System (DBMS) Optimizer to select their own indexes. We introduce a heuristic to guide the index selection process, which is implemented by a tool called AISIO that runs integrated to the DBMS. Our tool captures Structured Query Language (SQL) statements submitted to the DBMS, analyzes them and, based on the implemented heuristic and on the DBMS Optimizer statistics, recommends which indexes could potentially optimize the performance of those transactions. The tool also provides a measure of the performance gains obtained from the use of the recommended indexes. To illustrate our tool, we integrate it to the PostgreSQL DBMS and experimental results are presented to evidence our contributions.**

Keywords-index selection problem; database tuning;

I. INTRODUCTION

In spite of the significant effort that has been expended by Database Administrators (DBA) and researchers in the past few years, and of the clear progress that has been achieved, improving the performance of database transactions remains a challenge in practice. Much of this challenge can be directly associated to how data records are logically organized in a computational storage device and, especially, to how quickly those records can be accessed and processed.

In this sense, index structures have been associated to relational Database Management Systems (DBMS) as a way to allow data lines (tuples) to be more rapidly accessed [1]. This corresponds to a logical structure that maps the address where the data are stored physically.

Then, when a data is requested, the DBMS firstly checks the index structure, which is ordered and contains the physical data address, to only then retrieving the real data from the hard-drive, without further searching. This in general allows data records to be more quickly accessed and processed. The data structures most commonly used to represent indexes are balanced B+ trees, whose leaves contain a sequence of keypointer pairs, where keys are ordered by their values.

However, for a data index structure to be efficient in terms of performance, several aspects have to be considered. For example, the construction and configuration of indexes directly impact on their resulting performance. In some cases, the inappropriate use of indexes makes the approach useless and may even introduce unnecessary overhead to a data transaction.

In this context, the index selection problem has increasingly receiving attention of DBA and researchers. This activity is not straightforward and involves a number of factors, including to take into account the database system workload (input instructions), data model, etc., to only then properly selecting tables and columns to be indexed [2].

In the literature, some tools have been integrated to the DBMS to promote index selection [3], [4]. Most of these initiatives have commercial purpose, though, which prevents them to be extended for general purpose. Other approaches have suggested to take from the DBMS the responsibility for selecting indexes, letting this assignment to be done by DBMS-external tools [5]. This approach may lead a tool to calculate operation costs differently of the cost model implemented by the DBMS. As a consequence, the structure of indexes constructed by the tool may be inefficient, or it may be otherwise efficient but ignored by the DBMS optimizer, by understanding that it does not aggregate performance benefits.

In this paper, we show that better results can be obtained by letting the DBMS Optimizer itself internally manage the index selection. We present a heuristic to be associated to the DBMS, to guide the index selection process. The bridge between heuristic and DBMS is implemented by a tool named Automatic Index Selection Integrated into Optimizer (AISIO) that runs integrated to the DBMS. AISIO captures Structured Query Language (SQL) statements submitted to the DBMS, analyzes them and, based on the heuristic rules, statistical data and on outputs received from the DBMS Optimizer, recommends which indexes could potentially optimize the performance of those transactions. The tool also provides a measure of the performance gains resulting from the use of the recommended indexes. Finally, we integrate our approach to the PostgreSQL DBMS, and an example is presented in order to check its efficiency.

The paper is structurally organized as follows: Sec. II integration to a real DBMS; Sec. IV introduces the example, while conclusions and perspectives are discussed in Sec. V.

II. PROPOSAL FOR AUTOMATIC INDEX SELECTION

We describe in the following our proposal to automatically solve the index selection problem. The basic idea is to use the DBMS optimizer itself to find the best indexes for a given SQL statement, does not requiring, therefore, external tools to be associated to the DBMS. The tool we provide works combined to the DBMS.

Internally, the search for candidate indexes starts before initiating the optimization process. The tool, taking advantages from DBMS statistics and based on the heuristic to be presented next, enumerates relevant indexes to be potentially used. Then, it is created a hypothetical indexes list in the DBMS catalogue. When a SQL query is processed, the tool verifies which indexes have been selected by the DBMS optimizer to construct its query execution plan. If a hypothetical index is found in the execution plan, it is recommended for the DBA to be created. Performance estimations are also provided and the decision on whether or not to create the index is left in charge of the DBA.

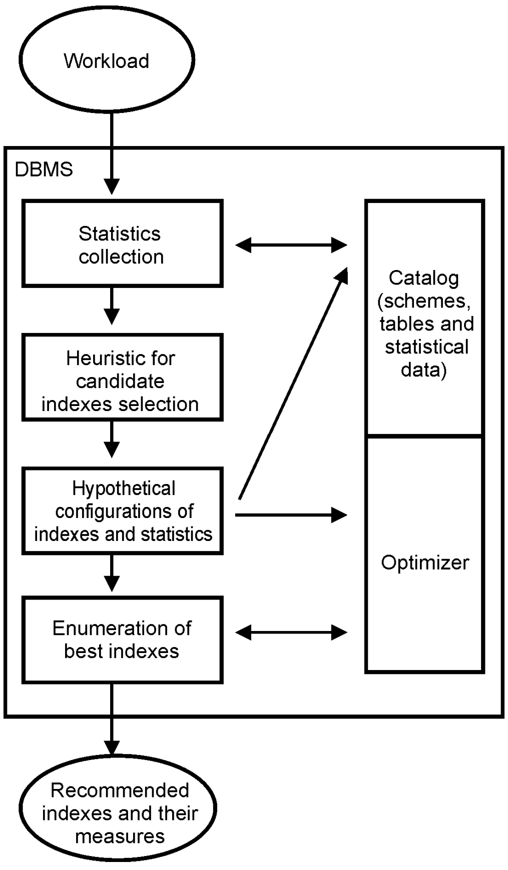
In the following we present the core of our approach. It is first presented the architecture of the implemented tool. Then, we describe a key step of this architecture: the heuristic proposed to guide the process of selecting candidate indexes. Afterwards, we discuss the main benefits of our approach and how it has been integrated to the PostgreSQL DBMS. Finally, an example illustrates our contributions.

A. *Tool Architecture*

To use our tool, the user must access any SQL client, and execute the appropriate command to start the AISIO tool. This procedure is described in the user guide, provided in [8]. Afterwards, it must be entered a workload, i.e., a SQL statement which it is intended to improve the performance. By doing this, the user is actually triggering a sequence of steps (see Figure 1) internally organized by our tool. After processing the informed SQL, it is shown an output containing the indexes recommended to the SQL statement submitted. Figure 1 shows the main modules of our tool, their relationship and how they flow through a cascade of stages.

The procedure for recommending indexes for the a SQL statement, according to the flow in Figure 1, involves the following seven steps:

1) *Workload:* this refers to the SQL statements supplied by the user as input.
2) *Statistics collection:* this is the stage when the optimizer is asked for statistical data on the submitted SQL statements.
3) *Heuristic:* in this stage, the SQL statement is scanned in order to find columns for which the creation of an index would be beneficial. The heuristic rules that guide this stage are detailed in Subsection II-B.
4) *Creation of hypothetical indexes:* candidates indexes, selected in the previous stage (Heuristic), are created in the DBMS, but with no index data. However, to allow the simulation of hypothetical configurations, the metadata of those indexes are recorded into the DBMS catalog, as if they were real indexes. This is done by having as a parameter the metadata of the table on which the index is linked.
5) *Optimization process:* with the hypothetical indexes created in the database scheme, the optimization process begins. This works similar to usual DBMS optimization procedures, but now real and hypothetical indexes are considered in the optimization process.
6) *Listing selected indexes and statistical data:* in order to explicit which indexes have been selected by the DBMS optimizer in the previous step, one can list the index appearing in the execution plan generated by the DBMS. Also, it is possible to list statistics of the hypothetical indexes that have been selected.
7) *Index recommendation and performance measure:* this last step generates the index-recommendation together with their statistical and cost-benefit data. These information compose the output to the user.



Workload
DBMS
Statistics collection
Heuristic for candidate indexes selection
Hypothetical configurations of indexes and statistics
Enumeration of best indexes
Catalog (schemes, tables and statistical data)
Optimizer
Recommended indexes and their measures

Figure 1. AISIO Tool architecture.

Figure 2 illustrates the index recommendation for a real SQL input. The heuristic that leads to the results in Figure 2 is explained next.

B. *Heuristic for selecting candidate indexes*

Given an input SQL statement, the choice for candidate indexes can be described by the four-step process that follows:

1) *Scanning of the SQL statement to identify single-column candidate indexes that satisfy the following criteria:*
   - *columns involving equal predicates;*
   - *columns involved in clauses ORDER BY, GROUP BY or join predicates;*
   - *columns appearing in interval restrictions;*
   - *columns appearing in other predicate (for example, like); and*
   - *other column composing the SQL statement.*
2) *Composition of single-column candidate indexes. This composition exponentially reproduces the index combinations;*
3) *Formation of multiple columns candidate indexes;*

```
$ aisio –c
"select i.type,i.address,l.value,l.location from property i, location l
 where value between 1000 and 3000 and initial_date >= '20130101'
and  i.id_property = l.id_property order by l.value;"

--- started execution 2013-10-02-14.01.23.51186
--- recommended indexes for this workload is ready.
--- Found set of 1 recommended index.

Cost actual: 9205.01
Cost optimized: 6690.31
Benefit: 2514.69
Global benefit: 0.41
Creation cost: 10602.05
Performance gain: 29.42%
Size: 6096 kb
Column(s): value, initial_date

--- SQL Statements of recommended indexes:
create index index_location on location
using btree(value, initial_date);

--- finished at timestamp: 2013-10-02-14.01.24.23579
```

Figure 2. Using the step-guide over an example

4) *Composition of the final list of candidate indexes to be used. This is done through the simple union of multiple column and single-column indexes;*

One important advantage of the proposed heuristic is that it exhaustively enumerates candidate indexes. Although this can be computationally expensive, in certain cases, for the proposed heuristic the operation looks feasible and potentially supplies the best choice for index selection. Further details on the heuristic are given in [6].

III. Integration with PostgreSQL

The integration between the presented architecture and heuristic, has been implemented by a tool named AISIO. The tool was developed in the Object-Relational DBMS PostgreSQL [7]. PostgreSQL has been chosen mainly because it is open-source and yet expressive for practical purposes. This makes both analysis and possible extensions of our approach easier and more applicable.

Regarding to implementation issues, we remark that the tool has been entirely programmed in language C. The choice is motivated by the fact that PostgreSQL is itself developed in C, so facilitating its integration with the tool. In its current version [8], the tool's code has been at all compiled together with the DBMS modules. Figure 3 outlines how AISIO is integrated to the DBMS PostgreSQL.

The communication model used by PostgreSQL is clientserver. A client-server architecture for DBMS works as a single process called Postmaster, encapsulating the server and receiving client connections. This process is responsible for allocating memory to be then shared among all the subprocesses managed by the DBMS, as well as for communicating these subprocesses into the server.

For every input command received from the user, the AISIO module is called to return the index recommendation for that SQL statement. So, the whole set of data that is managed by the DBMS core is also available for the AISIO core. This is one of the aspects that makes our approach advantageous with respect to the tool that work externally to the DBMS. Besides of integrating requests information between DBMS and tool, we also provide for the tool to access the DBMS catalogue, which contemplates records that can help with the creation of hypothetical index and their statistics.

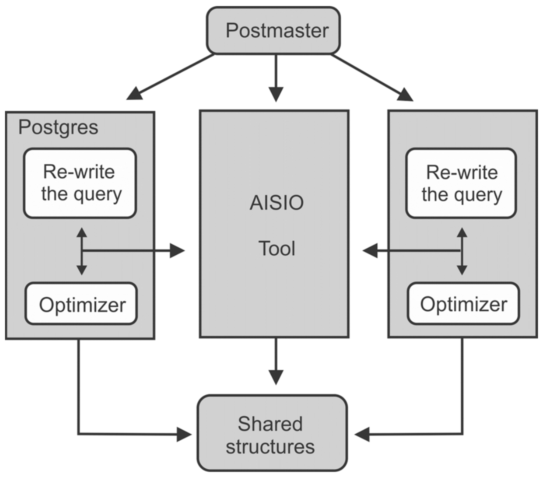


Figure 3. Integration of AISIO into the DBMS

IV. Performance Evaluation

A number of performance aspects must be considered when recommending index to be selected by a DBMS, including the quality of the recommended indexes, a measure of the real gain possible from their use, etc. Answering such questions is not usually straightforward. In the following, we provide a case study that allows to assess in more details how much helpful can be our tool for the index selection process.

To proceed with the experiments, we integrate the AISIO tool to the PostgreSQL 8.4, running on the platform Slackware Linux. To properly evaluate the efficiency of our tool, its estimations have been compared against results provided by the Open Source Development Labs (OSDL) DBT-2 toolkit [9]. This toolkit is provided by the Transaction Processing Performance Council (TPC) [10] and uses the TPC-C [10] benchmark to submit an OLTP workload to the PostgreSQL. The environment simulated by the benchmark consists of a transaction-processing system of a wholesale goods company. The company is geographically dispersed and sales are distributed from a number of warehouses and districts. The size of the organization is defined by its number of warehouses. As the company grows, more warehouses and districts are created and equivalently grows the benchmark scale factor.

The performance measure used by the benchmark is the tpmC, which measures the number of transactions of the type New Order, that are completed per minute. Although this measure is related to the throughput of only one class of transaction, other transactions have direct impact on the response time of New Order operations, thus ensuring that the system performance is tested as a whole. In order to evaluate the contributions brought by AISIO tool, we have collected the system throughput in three different scenarios:

1) *without any index in the database;*
2) *with index generated by DBT-2 toolkit; and*
3) *with index generated by AISIO tool.*

The results obtained for each scenario are depicted in Figure 4 and discussed afterwards.

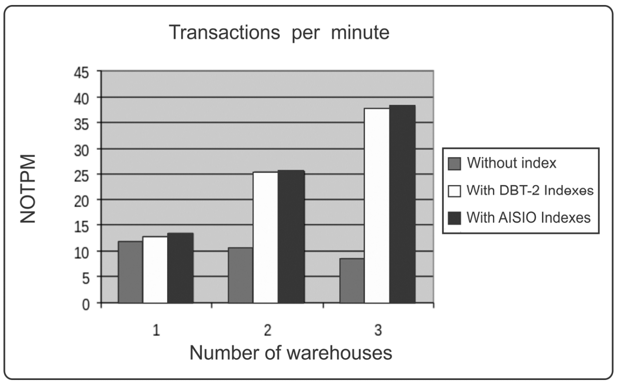


Figure 4. NOTPM (number of transactions per minute).

The first scenario leads to the smallest throughput in the benchmark. Since there is no indexing scheme, as the database increases with the growing number of warehouses, there is a constant fall-off in throughput. This is because in databases without indexes, requests are sequentially handled, so that more time is spent processing each request when the number of data records increases.

The second scenario uses indexes defined by the DBT-2 benchmark. Compared to the first scenario there is a significant performance (throughput) improvement. With indexes being now used, the increasing in the number of database records leads to more terminals accessing the DBMS and, as a consequence, to more transactions being processed per minute. When using the indexed DBT-2 benchmark, the response time of the transactions was noted to remain practically stabilized, independently of the database growth.

To finally evaluate the third scenario, we generate indexes using the AISIO tool. As expected, and similarly to the second scenario, in the presence of indexes the DBMS resulted in a much higher throughput than in the first scenario. Moreover, third scenario has evidenced a slight gain with respect to the second scenario, which indicates that a performance improvement has resulted from our approach with respect to the benchmark test.

Actually, the slight advantage of our approach with respect to the DBT-2 toolkit does not really mean a small gain, in practical terms. Remark that the set of indexes used in the second scenario has been chosen by experts in database tuning, certainly with deep knowledge of the particular database structure. However, it may not always be possible to have available professionals for each particular DBMS to be used, for several reasons, including money, expertise, etc. Besides, there may be cases for which a data indexing scheme is too complex for a human perception, making it unattractive to exclusively depend on DBMS administrators. This enforces even more the importance of having automatic options for index selection. Note that, in the third evaluation, the indexes have been exclusively chosen by AISIO tool, in an automatic process without human interference

V. Conclusion

This paper investigates the index selection problem for performance improvement in relational databases systems. A heuristic has been defined to choose candidate indexes and a tool named AISIO has been provided to implement this heuristic. AISIO leads to an automated answer to the index selection problem and it has shown to be advantageous with respect to other approaches found in the literature.

Initially, a heuristic has been defined to choose candidate indexes. This task consists of enumerating indexes with one or more columns. These indexes are separated and hypothetically considered for inclusion in the DBMS catalogue. The index enumeration explores a large number of possible combinations, which composes an important advantage of our approach. Once created hypothetical indexes, the optimization process starts, considering both the real and hypothetical sets of indexes in the DBMS. Then, the DBMS Optimizer selects which index would improve the performance of the processed transaction, and points it out to the DBMS administrator, who finally decides whether or not they should be created.

It is shown that using the DBMS Optimizer, combined with AISIO, to evaluate a set of candidate indexes is a better strategy than the use of any external tool. Furthermore, as AISIO is an DBMS-integrated tool, it allows to take advantage from the cost model provided by the DBMS optimizer, which is a sophisticated mathematical model that properly addresses how to make SQL statements optimization.

Perspectives of future work include to extend our proposal to involve Decision Support Systems. In those systems, database queries are more frequently received than insert and update operations. Therefore, the indexes selection process can be directed to those features, which potentially leads to a greater cost-benefit ratio. We also intend to further improve our approach by adding statistical calculations on hypothetical index to enhance their efficiency.

References

[1] F. Korth, A. Silberchatz, and S. Sudarshan, Database System Concepts, 5th ed. McGraw Hill, 2005.

[2] D. Shasha and P. Bonnet, Database Tuning: Principles, experiments, and troubleshooting techniques, 1st ed. Morgan Kaufmann, 2003.

[3] S. Agrawal, S. Chaudhuri, and V. R. Narasayya, "Automated selection of materialized views and indexes in sql databases," in Proceedings of the 26th International Conference on Very Large Data Bases, ser. VLDB'00. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2000, pp. 496–505.

[4] A. Skelley, "Db2 advisor: An optimizer smart enough to recommend its own indexes" in Proceedings of the 16th International Conference on Data Engineering, ser. ICDE '00. Washington, DC, USA: IEEE Comp. Society, 2000, pp. 101–110.

[5] S. Choenni, H. Blanken, and T. Chang, "On the selection of secondary index in relational databases," IEEE Data and Knowledge Engineering, 1993, pp. 207–233.

[6] G. W. Pedrozo, "Architecture for index selection in relational databases using cost-based approach the optimizer," Master's thesis, Dep. de Inform´atica, Universidade Federal do Paraná (UFPR), 2008, (in portuguese).

[7] PostgreSQL, 2013. Available: http://www.postgresql.org

[8] W. G. Pedrozo, AISIO-Tool: Automatic index selection integrated into optimizer, 2013. Available: http://201.89.227.155/aisio-tool

[9] Open Source Development Labs - OSDL, 2013. . Available: http://sourceforge.net/apps/mediawiki/osdldbt

[10] TPC-C benchmark, 2013. Available: http://www.tpc.org/tpcc