Thomas Kissinger, Hannes Voigt, Wolfgang Lehner

SMIX Live. A Self-Managing Index Infrastructure for Dynamic Workloads

Erstveröffentlichung in / First published in:

2012 IEEE 28th International Conference on Data Engineering, Arlington, 01.-05.04.2012. IEEE, S. 1225-1228. ISBN 978-0-7695-4747-3.

DOI: http://dx.doi.org/10.1109/ICDE.2012.9

Final edited form was published in "2012 IEEE 28th International Conference on Data Engineering. Arlington 2012", S. 1225-1228, ISBN 978-0-7695-4747-3
http://dx.doi.org/10.1109/ICDE.2012.9

SMIX Live - A Self-Managing Index Infrastructure for Dynamic Workloads

Thomas Kissinger, Hannes Voigt, Wolfgang Lehner

Database Technology Group
Dresden University of Technology
01062 Dresden, Germany
{firstname.lastname}@tu-dresden.de

***Abstract*— As databases accumulate growing amounts of data at an increasing rate, *adaptive indexing* becomes more and more important. At the same time, applications and their use get more agile and flexible, resulting in less steady and less predictable workload characteristics. Being inert and coarse-grained, state-of-the-art index tuning techniques become less useful in such environments. Especially the full-column indexing paradigm results in lot of indexed but never queried data and prohibitively high memory and maintenance costs. In our demonstration, we present *Self-Managing Indexes*, a novel, adaptive, fine-grained, autonomous indexing infrastructure. In its core, our approach builds on a novel access path that automatically collects useful index information, discards useless index information, and competes with its kind for resources to host its index information. Compared to existing technologies for adaptive indexing, we are able to dynamically grow and shrink our indexes, instead of incrementally enhancing the index granularity. In the demonstration, we visualize performance and system measures for different scenarios and allow the user to interactively change several system parameters.**

I. Introduction

Indexes are the most fundamental technique to speed up queries on datasets. An index stores references to tuples in a special access path that allows fast querying of these indexed elements of the dataset. Database systems leverage indexes heavily to efficiently manage huge datasets. Since indexes support only fractions of a database's workload while requiring resources to be stored and maintained, indexing needs to be optimized. With changing data and shifting workloads, the optimum is a moving target. Index information that is beneficial today may be unprofitable tomorrow. Self-managing indexing, where index optimization is an integral part of the database system, is the way to permanently relieve the user from the burden of index optimization.

In relational databases, columns form the primary granularity of indexing; each index covers all values of one or more columns. As databases accumulate growing amounts of data at an increasing rate, the core problems of full column indexing become more evident. If the data in a column doubles over time, a full column index takes nearly twice the time to be created and consumes about twice the storage. At the same time the data of interest is unlikely to double in size as an increasing share of data is primarily kept for reasons of revision, linage, and versioning. In short, the trend of growing data sizes has two consequences: (1) The traditional full-column index will soon encompass and maintain mainly unused index information. (2) The traditional index tuning based on creating and dropping indexes will soon become prohibitively expensive.

The historic dominance of coarse-grained full column indexing has two practical reasons: (1) The user can define easily what should be indexed. (2) The query optimizer can determine easily which indexes apply to a query. However, these two reasons disappear when we consequently rethink indexing as a self-managing component in the database system. First, with self-managing indexing the task of index selection is taken away from the user. The system decides on its own. Second, the optimizer's need to find applicable indexes arises only because indexes are mainly secondary, optional, user-created access-paths. Once indexing is directly integrated in the system as a default access path available on every column, it is not required to track which parts of data are indexed or not. Self-managing indexing not only disburdens the user from index optimization tasks, it also paves the way to more fine-grained indexing.

In this demonstration, we present a novel, adaptable, fine-grained, autonomous indexing infrastructure for row stores, structuring rows to blocks or pages. It is based on the Self-Managing Index (SMIX), a new access path that automatically creates a partial index of the column it is working on. The SMIX indexing infrastructure comes with only a very few configuration knobs, mainly the amount of resources that can be used for indexing. SMIXs distribute the heavy lifting of index creation over time and focus index creation on the data of interest. Furthermore, the approach does not involve expensive what-if calls to the query optimizer. This way, SMIXs reduce the required user interaction dramatically without sacrificing performance by missing indexing opportunities or imposing to much overhead on the DBS.

II. Related Work

Substantial research has been done in the field of index tuning. First research in that area dates back to the late 1970s. Nowadays commercial database management systems offer index tuning tools, [1], [2], [3], which recommend an index configuration for a given workload and a storage bound the configuration has to fit into. However, all these state-of-the-art tools consider the database workload as static and predictable.

Final edited form was published in "2012 IEEE 28th International Conference on Data Engineering. Arlington 2012", S. 1225-1228, ISBN 978-0-7695-4747-3
http://dx.doi.org/10.1109/ICDE.2012.9

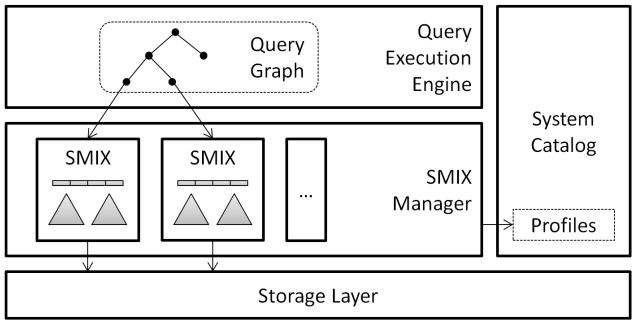

Fig. 1. SMIX in Database Architecture

Other approaches extend the idea of the index tuning tool to dynamic workloads. Here, the tool analyses the workload as a series of events over time and recommends a series of index configurations [4], [5]. Since all index tuning tools work offline, the user has to be able to predict the regular workload.

Research concentrated on autonomous index tuning in the recent past. A couple of solutions have been proposed [6], [7], [8]. All of them stick with the core concepts of the index tuning tools: full-column indexing and expensive query cost evaluation. Consequently, they suffer from the same two drawbacks: (1) The index tuning remains very coarse-grained and the resulting indexes are likely to include a lot of data that is not of interest. (2) The index tuning requires expensive creation and dropping of complete indexes. Although SMIXs are also an autonomous index tuning approach, they are fundamentally different. SMIXs inherently index only data of interest and build on less expensive incremental index creation and adaption.

Approaches based on incremental partitioning of unsorted data [9], [10] (also known as database cracking) and incremental merging of pre-sorted data chunks [11], [12] specifically remedy the creation costs of indexes. Both concepts piggyback on queries to create index information on the tuples that are requested; this lowers the creation costs and distributes the effort over time. Although approaches are appealing, they do not represent solutions to the index optimization problem. Without any dropping of index information, every incremental index creation converges to a regular full index so that also uninteresting tuples will be indexed at some point. Our SMIX approach is more comprehensive. SMIXs do not only involve incremental collection of index information, but also incremental displacement. Instead of converging to full indexation, SMIXs converge to the workload.

III. System Overview

The core idea of SMIX is a new default access path that adapts itself to the workload. This requires two novel components to the database architecture shown in Figure 1. The first component is the SMIX itself, the new self-managing access path. The second component is the SMIX manager, which supervises the SMIX population in the system.

A SMIX combines the abilities of traditional table scan and index scan in a single access path. Like a traditional table scan, a SMIX acts as a default built-in access path, which is available on every column and does not have to be created explicitly. Like a traditional index, a SMIX incorporates index information, which allows reducing page accesses significantly. Implementation-wise, a SMIX even reuse the logic of these traditional access paths. A SMIX autonomously collects index information based on the tuples that are accessed by the workload. It directly leverages this collected information for the next accesses, even if these accesses relate to other tuples. Additionally, a SMIX not only collects new index information, a SMIX also discards index information that turned out to be less useful. Therefore, a SMIX adapts to the data workload and is also able to control its use of storage and memory resources.

SMIXs co-exist to traditional access paths in the system. The query optimizer still decides which access path to take for a specific query. It applies two general rules for the access path selection: (1) It always chooses a SMIX scan over a table scan, if the optimizer would take an covering index on this column, because a SMIX can quickly adapt to better performance. (2) It always chooses a traditional index scan over the SMIX scan if a single column index is present, because a SMIX rarely exceeds the performance of a traditional index scan and redundant index information should be avoided. If multicolumn indexes are present on the queried column, the optimizer relies on traditional statistics-based decision rules. In order to accomplish that, every SMIX maintains statistics about itself in the system catalog, similar to traditional index and table statistics.

SMIXs are query-driven; they are not created explicitly. A SMIX that was never accessed, does not consume any space. Each indexable column has a catalog entry indicating if it has an initialized SMIX present. A query can utilize a SMIX scan on a column even if the column's SMIX has not been initialized. The first SMIX scan on a column will initialize the SMIX on that column.

Considering a query against a column c, a partial index IX on c can help answering the query in two cases:

1) If all tuples with value x in column c are indexed in index IX, the tuples answering a query $c = x$ can be directly discovered with IX. This is the desired case, because a table scan can be avoided. The more queries fall into this category, the better the partial index is adapted to the workload.
2) If all tuples in a page p are indexed in index IX, all tuples matching a query on c can be discovered by a table scan. This table scan is able to skip p, but needs to combine its result set with the output of an IX index scan. Hence, skipping a lot of pages during a table scan helps to mitigate query execution costs in situations where the partial index is not well adapted to the workload and many table scans are necessary to answer queries.

To cover both cases, a SMIX collects index information in two separate B-Trees.

Final edited form was published in "2012 IEEE 28th International Conference on Data Engineering. Arlington 2012", S. 1225-1228, ISBN 978-0-7695-4747-3
http://dx.doi.org/10.1109/ICDE.2012.9

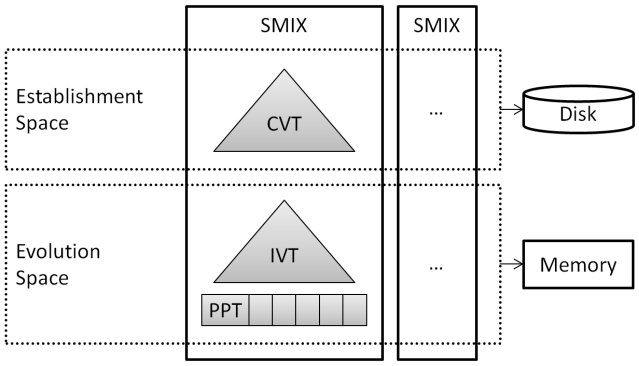

Fig. 2. SMIX in index spaces

1) The *covered values tree* (CVT) indexes the most queried values to answer most of the common queries efficiently.
2) The *intermediate values tree* (IVT) completes the indexation of a subset of pages, to speed up table scans.

This way, a SMIX is able to adapt to a workload while quickly leveraging collected index information.

Besides CVT and IVT, a SMIX manages a list of counters, called *page population table* (PPT). The PPT contains a counter for each page of the table the SMIX serves. Each counter shows the number of tuples in its page that are indexed neither in the CVT nor in the IVT. The PPT helps to quickly select pages that are most worthwhile to be indexed in the IVT (pages with a low counter greater than zero). It also allows to easily identify pages that can be skipped by a table scan (pages with a counter equal to zero). A SMIX initializes the PPT with the first table scan it has to perform and subsequently maintains the PPT incrementally.

The SMIX access path automatically collects index information on potentially every column. Two principles keep the SMIX population of a database from exceeding a configurable global resource limit.

1) Every SMIX has an individual resource quota and it is able to displace less queried index entries to lower its resource usage.
2) All SMIXs compete for the globally granted resources, so that invaluable index information automatically drops out of the system.

The globally available resources for indexing are index spaces, where the index information is stored. For SMIXs, we distinguish two types of index spaces.

1) The *establishment space* represents disk resources; it offers persistency and supports crash recovery. A SMIX stores its CVT in the establishment space, since the CVT has proven value for the workload.
2) The *evolution space* represents faster main memory resources; it is transient and does not support crash recovery. A SMIX stores its IVT and PPT int the evolution space. Both are supporting structures and contain less valuable indexing information.

Figure 2 summarizes this setup. Especially when a SMIX is barely adapted to the workload, it makes heavily use of the IVT. Hence, the main-memory-based evolution space allows a faster adaption at lower costs compared to disk.

The SMIX manager is the supervisor component for all SMIXs in the system. Since SMIXs are automatically created and allocate new storage and memory resource on their own, they need to be controlled, in order to not exceed the globally available resources. The SMIX manager collects access statistics for every SMIX. Based on these statistics, the SMIX manager assigns quotas for establishment space and evolution space to each SMIX, while the absolute size of establishment space and evolution space is configured by the DBA. To enforce the quotas, the SMIX manager orders displacement of index information. In evolution space, a SMIX simply discards the complete IVT to free space. In the establishment space, a SMIX takes more care and removes only single leaf nodes of the CVT. Additionally, a SMIX removes CVT leaf nodes proactively if they are barely accessed.

IV. Demonstration

In this demo, we will show SMIXs in action. We have implemented SMIX in the open source database system PostgreSQL. The core presentation and interaction element of the demo is a GUI client called *SMIX Live dashboard*, which allows controlling and monitoring the SMIX implementation

With SMIX Live dashboard, the SMIX operating can be shown in detail for various parameter settings and workload scenarios. A single SMIX demo run involves setting the technical parameters of the SMIX concept, generating a workload with specific characteristics, and run generated workload with the SMIX implementation. The SMIX dashboard allows controlling all these step handily. While executing the workload, the SMIX Live dashboard monitors runtime measures and displays them to reveal the inner operations and performance of the SMIX concept to the demo visitors. All demo workloads build on synthetic scenarios. This allows controlling all aspects of a run.

We have a number of prepared SMIX demo runs, to give an introduction to demo visitors. The prepared demo runs are designed to show the core aspects of the SMIX concept. However, the demo is designed to be interactive. Demo visitors are invited to suggest different settings and scenarios. In general, which particular settings and scenarios are shown is guided by the conversation with the demo visitor, so that it fits best the visitor's way of approaching and understanding the SMIX concept.

Figure 3 shows the SMIX dashboard in detail. The SMIX dashboard is divided in three main parts:

1) the scenario selection
2) the SMIX parameter configuration panel
3) the measurement area.

In the following, we describe these parts more detailed.

Scenario Selection: The scenario selection allows the visitor to choose from five predefined scenarios. Each scenario runs a different workload on our SMIX implementation. The first three scenarios involve a single SMIX, each with another

Final edited form was published in "2012 IEEE 28th International Conference on Data Engineering. Arlington 2012", S. 1225-1228, ISBN 978-0-7695-4747-3
http://dx.doi.org/10.1109/ICDE.2012.9

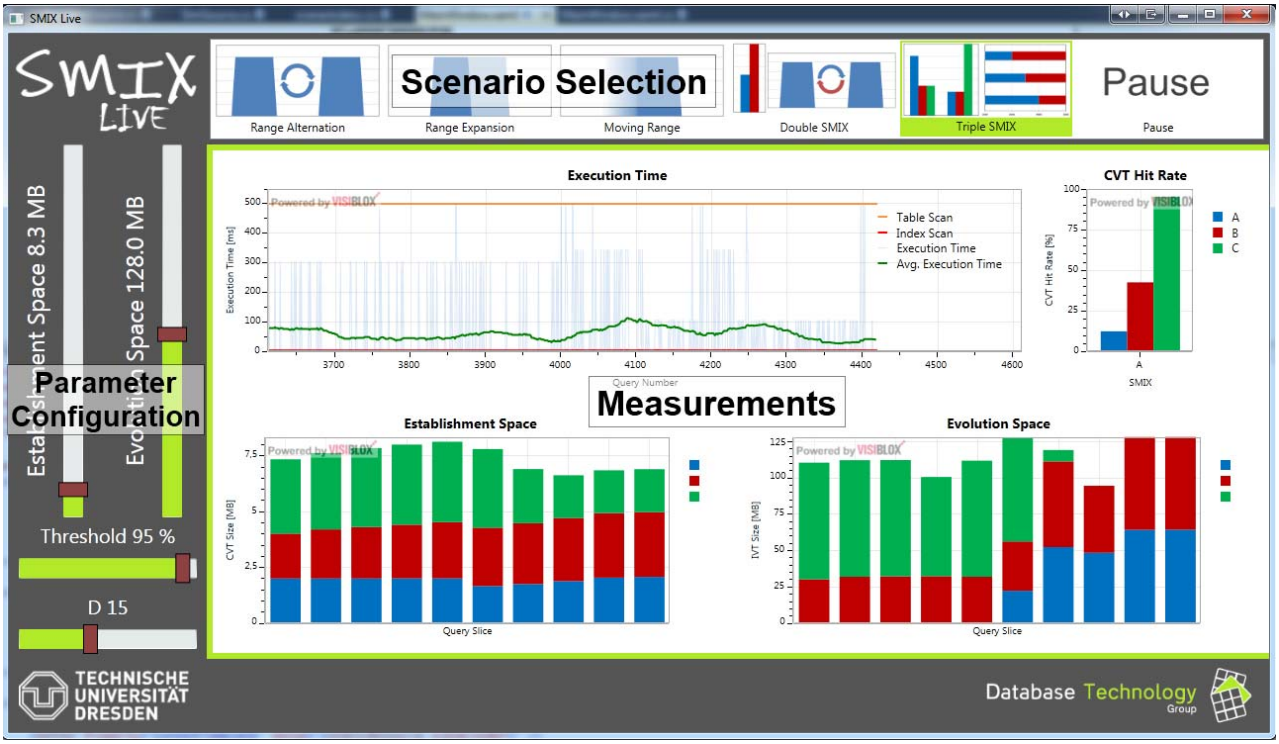

Fig. 3. SMIX Live Dashboard

workload pattern that changes over time. This helps the demo visitor to easily understand the SMIX indexing strategy. We provide workload patterns for:

1) instantly changing query ranges
2) expanding query ranges
3) slowly moving query ranges.

A query range means a continuous domain of values, on which the queries are executed. The remaining two scenarios generate workloads that involve multiple SMIXs. This gives an understanding, how the SMIX manager distributes the available storage between the individual SMIXs over time. Each of this complex scenarios uses changing column access frequencies and query ranges.

SMIX Parameter Configuration Panel: This Panel allows the demo visitor to change the technical parameters of the SMIX concept. All parameter changes are applied live to the SMIX implementation, which shows immediate response. We allow the core parameters: (1) size of the establishment and evolution space, (2) stability threshold, and (3) automatic displacement aggressiveness, to be changed. While playing with these parameters, the visitor will understand the influence of each of them.

Measurement Area: The measurement area updates frequently and shows the query performance as well as the internal SMIX state. We included live charts for (1) the query execution time (with the table scan and index scan as baseline), (2) the CVT hit rate, and (3) the usage of the establishment and evolution space. With the help of those charts, we are able to describe SMIX behavior under certain circumstances to the visitor and suspend the scenario execution in case of deeper questions.

All these possibilities of the SMIX Live dashboard allow visitors to immerse theirself in the SMIX concept.

References

[1] S. Agrawal, S. Chaudhuri, L. Kollár, A. P. Marathe, V. R. Narasayya, and M. Syamala, "Database Tuning Advisor for Microsoft SQL Server 2005." in *VLDB'04*, 2004.
[2] D. C. Zilio, J. Rao, S. Lightstone, G. M. Lohman, A. Storm, C. Garcia-Arellano, and S. Fadden, "DB2 Design Advisor: Integrated automatic physical database design." in *VLDB'04*, 2004.
[3] B. Dageville, D. Das, K. Dias, K. Yagoub, M. Zaït, and M. Ziauddin, "Automatic SQL tuning in Oracle 10g," in *VLDB'04*, 2004.
[4] S. Agrawal, E. Chu, and V. R. Narasayya, "Automatic physical database tuning: Workload as a sequence." in *SIGMOD'06*, 2006.
[5] H. Voigt, W. Lehner, and K. Salem, "Constrained dynamic physical database design," in *SMDB'08*, 2008.
[6] N. Bruno and S. Chaudhuri, "An online approach to physical design tuning." in *ICDE'07*, 2007.
[7] K. Schnaitter, S. Abiteboul, T. Milo, and N. Polyzotis, "On-line index selection for shifting workloads," in *SMDB'07*, 2007.
[8] K.-U. Sattler, M. Luehring, K. Schmidt, and E. Schallehn, "Autonomous management of soft indexes," in *SMDB'07*, 2007.
[9] M. L. Kersten and S. Manegold, "Cracking the database store," in *CIDR'05*, 2005.
[10] S. Idreos, M. L. Kersten, and S. Manegold, "Database cracking," in *CIDR'07*, 2007.
[11] G. Graefe and H. A. Kuno, "Self-selecting, self-tuning, incrementally optimized indexes," in *EDBT'10*, vol. 426, 2010.
[12] ——, "Adaptive indexing for relational keys," in *ICDEW'10*, 2010.