Adaptive Indexing in Modern Database Kernels

Stratos Idreos
CWI Amsterdam
idreos@cwi.nl

Stefan Manegold
CWI Amsterdam
manegold@cwi.nl

Goetz Graefe
HP Labs, Palo Alto
goetz.graefe@hp.com

ABSTRACT

Physical design represents one of the hardest problems for database management systems. Without proper tuning, systems cannot achieve good performance. Offline indexing creates indexes a priori assuming good workload knowledge and idle time. More recently, online indexing monitors the workload trends and creates or drops indexes online. Adaptive indexing takes another step towards completely automating the tuning process of a database system, by enabling incremental and partial online indexing. The main idea is that physical design changes continuously, adaptively, partially, incrementally and on demand while processing queries as part of the execution operators. As such it brings a plethora of opportunities for rethinking and improving every single corner of database system design.

We will analyze the indexing space between offline, online and adaptive indexing through several state of the art indexing techniques, e.g., what-if analysis and soft indexes. We will discuss in detail adaptive indexing techniques such as database cracking, adaptive merging, sideways cracking and various hybrids that try to balance the online tuning overhead with the convergence speed to optimal performance. In addition, we will discuss how various aspects of modern techniques for database architectures, such as vectorization, bulk processing, column-store execution and storage affect adaptive indexing. Finally, we will discuss several open research topics towards fully automomous database kernels.

1. INTRODUCTION

Physical Design. Physical design is a mandatory step in order to extract good performance from a database system. Over the years, this tuning step became harder and harder. As the workloads became more complex and the database systems got more knobs and became more sensitive to their set-up, the tuning phase has outgrown the capabilities of database administrators. Now, we rely on auto-tuning tools to assist database administrators and to make good tuning possible. Contemporary index selection tools rely on monitoring database requests and their execution plans, occasionally invoking creation or removal of indexes on tables and views.

Dynamic Workloads. The combination of auto-tuning tools and administrators works well for rather stable environments with accurate workload knowledge and plenty of a priori slack time to invest in physical design. However, in the context of dynamic workloads, such tools tend to suffer from the following three weaknesses. First, the interval between monitoring and index creation can exceed the duration of a specific request pattern, in which case there is no benefit to those tools. Second, even if that is not the case, there is no index support during this interval. Data access during the monitoring interval neither benefits from nor aids index creation efforts, and eventual index creation imposes an additional load that interferes with query execution. Last, but not least, traditional indexes on tables cover all rows equally, even if some rows are needed often and some never.

Dynamic Indexing. Dynamic and unpredictable workloads require non static approaches to indexing. *Online* indexing is a new approach that tries to take all indexing decisions on-line, i.e., as the workload evolves and as we understand better its shape and the requirements. The system *continuously* adapts, i.e., continuously monitors the performance, drops and creates indexes when this is considered beneficial.

Creating indexes on-the-fly though, may penalize individual queries as well as it may take time until we understand a new pattern is emerging. *Adaptive* indexing removes these issues by enabling efficient *incremental and partial* online indexing while *instantly* reacting to workload changes. The main innovation is the ability to build indices incrementally as part of query execution; as queries arrive, the actual query operators physically reorganize data to match the workload.

One of the main "rules" of adaptive indexing is the following.

Every query is treated as an advice of how data should be stored.

In this way, the actual storage continuously adapts and access patterns improve as the workload changes. No human intervention is required and everything happens in a transparent way to the user.

Tutorial Plan. In this tutorial, we will first present the research space between adaptive indexing and the more traditional offline indexing as well as the latest efforts on online indexing. We will distinguish the application scenario for each case and the challenges involved. We will motivate the new research path of adaptive indexing as well as the need to combine all three areas towards a new holistic tuning method.

We will discuss in detail several state of the art offline, online and adaptive indexing techniques. We will see the basic approaches to what-if analysis for offline indexing as well as modern techniques for online indexing, such as soft indexes. Database cracking, sideways cracking, partial cracking, adaptive merging as well as various hybrid adaptive indexing methods will be covered in detail. We will discuss the first adaptive kernel over the open source column-store MonetDB and how this compares with database architectures that support offline and online analysis. In addition, we will discuss several optimization issues such as maintenance of adaptive indexing structures, improving convergence speed, etc. Furthermore, updates, concurrency control, and adaptive indexing for several database operators such as joins, selects and tuple reconstruction will be discussed.

Finally, we will discuss several open topics in the context of adaptive and dynamic indexing and how these can affect database research and the database community at large. For example, we will discuss how adaptive indexing can be incorporated in traditional systems, in the form of row-store and pipelined processing, b-trees, etc.

2. TUTORIAL MATERIAL

Physical Design. The tutorial will start by describing the physical design problem; the importance of creating the proper indexes as well as the ingredients for successful physical design.

Offline Analysis. We will discuss in detail the concept of offline analysis as it exists in every major database product, i.e., we will discuss the what-if analysis paradigm and how modern auto-tuning tools work and interact with the database system, e.g., [4, 6, 7, 11, 1, 5, 20, 21]. Such offline approaches to runtime index tuning are non-adaptive, meaning that index tuning and query processing operations are distinct from each other. These approaches first analyze a given sample workload and then decide which indexes to create or drop based on the observations. Both index tuning and index creation costs may impact the actual database workload. Once a decision is made, it affects all key ranges in an index. We will describe in detail the benefits as well as the bottlenecks such an approach can have for dynamic environments.

Online Analysis. Online analysis attacks the problem of dynamic workloads where any offline decisions might soon become invalid and new indexes are required. A number of recent efforts propose variations of online indexing, e.g., [3, 18, 2, 16]. The general idea is that the basic concepts of offline analysis are transferred online, i.e., while processing queries the system monitors the workload and the performance, it questions the need for different indexes and once certain thresholds are passed it triggers the creation of new indexes and possibly it drops old ones. We will discuss in detail this approach and the benefits it brings as well as the bottlenecks it creates by penalizing certain queries with high overhead.

Adaptive Indexing. Once the concepts of offline and online analysis for physical design are clear, we are ready to move onto the adaptive indexing discussion. The bottlenecks of offline and online analysis for dynamic workloads motivate adaptive indexing. We will discuss in detail the scenarios where each approach is applicable and the benefits of adaptive indexing as shown in [12, 13, 14, 15, 9, 10, 8]. Adaptive indexing aims to enable incremental, efficient adaptive indexing, i.e., index creation and optimization as side effects of query execution, with the implicit benefit that only tables, columns, and key ranges truly queried are optimized. The more often a key range is queried, the more its representation is optimized. Columns that are not queried are not indexed, and key ranges that are not queried are not optimized. Overhead for incremental index creation is minimal, and disappears when a range has been fully-optimized.

Database Cracking. We will start with a discussion on database cracking [12, 13, 14]. Database cracking introduces the notion of incrementally building indexes as a byproduct of operator calls during query processing. It relies on an extremely lazy approach in order to introduce minimal overhead to individual queries. The main idea is that data is continuously physically reorganized to match the workload.

Selection Cracking. We will first discuss selection cracking [12], where only the select operator is responsible for adaptive indexing steps. Here, the idea is that in order to perform a simple selection, the respective column needs to be reorganized to collect all qualifying values in a contiguous area. This results in a continuous injection of more and more structure knowledge that future queries can exploit. We will discuss in detail the first complete prototype and its performance when exploiting these techniques.

Column-Stores. The database cracking work exploits and in fact relies on several column-store properties, such as storage on fixed width dense arrays, bulk processing and late tuple reconstruction. We will take a few minutes in describing the basic concepts of modern column-store processing and why these features assist database cracking.

Cracking Updates. Then, we will discuss about updates [13] which are performed in the same adaptive philosophy. Updates are applied on demand, adaptively and incrementally during query processing and while cracking the data-

base arrays. We will show which are the main techniques and tradeoffs for updates as well as we will discuss open issues in this context.

Sideways Cracking. After that, a discussion on partial and sideways cracking [14] will show how to answer complex queries, e.g., TPC-H, with adaptive indexing while also taking into account storage bounds. We will discuss techniques that transfer cracking across multiple columns in a table as well as techniques that allow for partial and incremental materialization of the auxiliary cracking structures. In addition, we will discuss several optimization options in this context, such as adaptive alignment, new query plans and operators necessary, etc.

Adaptive Merging. Furthermore, we will discuss about adaptive merging [9, 10]. Adaptive merging, inspired by DB cracking, follows the same principles of continuous adaptation but introduces a crucial new idea, i.e., that of creating more active reactions to the workload. This is very useful, especially in disk based environments and it improves the convergence to the optimal performance of a full index.

Performance Metrics and Benchmark. As proposed in [8], two measures are crucial to characterize how quickly and efficiently a technique adapts index structures to a dynamic workload. These are: (1) the initialization cost incurred by the first query and (2) the number of queries that must be processed before a random query benefits from the index structure without incurring any overhead. We will discuss the first benchmark proposed for adaptive indexing [8] and how conclusions and performance in such a benchmark may affect the choice of indexing strategies.

Hybrid Adaptive Indexing Algorithms. Then, we will discuss the broad space of adaptive indexing as it is defined in [15] where multiple techniques are discussed on how to balance between initialization and convergence and how various design choices affect these parameters. We will begin with a comparison of the basic reorganization techniques of adaptive merging and database cracking in the context of column-stores and then we will proceed to discuss various alternative reorganization strategies towards the ideal adaptive indexing strategy.

Auto-tuning Kernels. Another important part of the discussion will be the requirements in order to apply adaptive indexing in a database kernel. We will discuss what it took to create the first adaptive indexing system as an extension of MonetDB and how other modern database kernels can adopt similar strategies. We will discuss in detail about changes necessary at all levels, e.g., operators, query plans, optimizer rules and various optimization and system dependent choices.

Prior Approaches. With the main part of adaptive indexing covered, we will take a step back and revisit the topic of why adaptive indexing brings something radically new to database research but also how it was inspired by past efforts that made the first steps towards partial and online indexing.

In the past, the recognition that some data items are more heavily queried than others has already led to the concept of partial indexes [17, 19]. A generalization is the concept of materialized views. Adaptive indexing though proposes a way to *incrementally* and adaptively create those indexes as part of query processing.

In addition, *soft indexes* is another recent interesting approach in the path of adaptive indexing [16]. Like monitor-and-tune approaches, soft indexes continually collects statistic for recommended indexes and then periodically and automatically solve the index selection problem. Like adaptive indexing, recommended indexes are generated (or dropped) during query processing. Unlike adaptive indexing, however, neither index recommendation nor creation is incremental; explicit statistics are kept and each recommended index is created and optimized to completion, although the command might be deferred. In addition, soft indexes simply exploit the scan of the relevant data, e.g., from a select operator and send this data to the index creation routine at the same time. On the contrary, database cracking overloads database operators with new algorithms that both answer the relevant operator and introduce small physical reorganization actions which makes lightweight adaptation possible.

In general, adaptive indexing and approaches that monitor queries then build indexes are mutually compatible. Policies established by the observe-and-tune techniques could provide information about the benefit and importance of different indexes, and adaptive indexing mechanisms could then create and refine the recommended index structures while minimizing additional workload.

Open Topics. The final part of the tutorial will present multiple open research topics for adaptive indexing. These range from concurrency control, disk based processing, long term maintenance of structures, as well as a future system that learns from all adaptive indexing, offline indexing and online indexing to create a database system that continuously and efficiently adapts to its environment. There are a plethora of open topics on almost all database design points which we believe will appeal to a broad audience. For several of this topics, we will discuss in detail their importance in the course of adaptive indexing, and possible paths towards solutions.

Duration. To cover all material discussed in this section, a 3 hours slot will be required. Alternatively, we can remove part of the background discussions and part of the detailed discussions to fit in a 1.5 hour slot.

3. TARGETED AUDIENCE

This tutorial targets database researchers from multiple fields.

In particular, it will appeal to core database architecture designers as adaptive indexing proposes a new way to design modern database kernels. It brings a new requirements and opportunities for database architecture design which involves new operators, algorithms, structures, query plans, etc.

Naturally, database researchers in the field of physical design, both offline and online, will be a perfect match for this tutorial as adaptive indexing complements these research

lines and brings a viable way of how to combine them.

In addition, researchers working on query optimization in general will fit in this tutorial as adaptive indexing brings numerous new opportunities for query optimization.

Finally, we believe the tutorial would be interesting for anyone working on adaptive techniques for any kind of query/data problem. The core ideas of adaptive indexing extend beyond the strict limits of database architectures, e.g., to how we can have adaptive data structures.

4. SHORT BIOGRAPHY

Stratos Idreos holds a tenure track senior researcher position with CWI, the Dutch National Research Center for Mathematics and Computer Science. The main focus of his research is on adaptive query processing and database architectures, mainly in the context of column-stores. He also works on stream processing, distributed query processing and scientific databases. Idreos obtained his PhD from CWI and University of Amsterdam. In the past he has also been with the Technical University of Crete, Greece, and held research internship positions with Microsoft Research, Redmond, with EPFL, Switzerland and with IBM Almaden. Idreos won the 2011 ACM SIGMOD Jim Gray Doctoral Dissertation award for his thesis on database cracking while in 2010 he was named a "Distinguished Scientist Excelling in Research abroad" by the Hellenic Ministry of National Defense.
Personal web page: `http://homepages.cwi.nl/~idreos/`

Stefan Manegold is the group leader of the database architecture research group at CWI in Amsterdam, The Netherlands. He received his PhD from the University of Amsterdam, The Netherlands, in 2002 and his Master (Diplom) in computer science from the Technical University of Clausthal, Germany, in 1994. Manegold's research work comprises database architectures, query processing algorithms and data management on modern hardware, as well as leveraging column-store database technology for efficient and scalable XML / XQuery processing, with a particular focus on optimization, performance, benchmarking and testing. Manegold co-authored more than 40 scientific publications, and recently received the VLDB 2009 10-year Best Paper Award together with his co-authors Peter Boncz and Martin Kersten. Stefan Manegold is a core member of the developers team of the open-source column-oriented database system MonetDB, co-founder of the DaMoN workshop series (co-located with SIGMOD since 2005), and co-chair of the Repeatability and Workability Evaluation for SIGMOD 2009 and 2010.
Personal web page: `http://homepages.cwi.nl/~manegold/`

Goetz Graefe is a HP Fellow researching database issues, primarily transactional indexing and robust query processing. Various database products employ techniques from his Exodus, Volcano, Cascades research projects. His best known works are in-depth surveys on query execution, sorting, and B-tree indexing.

5. REFERENCES

[1] S. Agrawal et al. Database Tuning Advisor for Microsoft SQL Server. VLDB 2004.

[2] N. Bruno and S. Chaudhuri. To Tune or not to Tune? A Lightweight Physical Design Alerter. VLDB 2006.

[3] N. Bruno and S. Chaudhuri. An online approach to physical design tuning. In *ICDE*, 2007.

[4] N. Bruno and S. Chaudhuri. Physical design refinement: the 'merge-reduce' approach. *ACM TODS*, 32(4):28:1–28:41, 2007.

[5] S. Chaudhuri and V. Narasayya. An efficient cost-driven index selection tool for Microsoft SQL Server. VLDB. 1997.

[6] S. Chaudhuri and V. R. Narasayya. Self-tuning database systems: A decade of progress. *VLDB*, pages 3–14, 2007.

[7] S. J. Finkelstein, M. Schkolnick, and P. Tiberio. Physical database design for relational databases. *ACM TODS*, 13(1):91–128, 1988.

[8] G. Graefe, S. Idreos, H. Kuno, and S. Manegold. Benchmarking adaptive indexing. *TPCTC*, pages 169–184, 2010.

[9] G. Graefe and H. Kuno. Adaptive indexing for relational keys. *SMDB*, pages 69–74, 2010.

[10] G. Graefe and H. Kuno. Self-selecting, self-tuning, incrementally optimized indexes. *EDBT*, pages 371–381, 2010.

[11] T. Härder. Selecting an optimal set of secondary indices. *Lecture Notes in Computer Science*, 44:146–160, 1976.

[12] S. Idreos, M. L. Kersten, and S. Manegold. Database cracking. *CIDR*, pages 68–78, 2007.

[13] S. Idreos, M. L. Kersten, and S. Manegold. Updating a cracked database. *SIGMOD*, pages 413–424, 2007.

[14] S. Idreos, M. L. Kersten, and S. Manegold. Self-organizing tuple reconstruction in column stores. *SIGMOD*, pages 297–308, 2009.

[15] S. Idreos, S. Manegold, H. Kuno, and G. Graefe. Merging what's cracked, cracking what's merged: Adaptive indexing in main-memory column-stores. *PVLDB*, 2011.

[16] M. Lühring, K.-U. Sattler, K. Schmidt, and E. Schallehn. Autonomous management of soft indexes. *SMDB*, pages 450–458, 2007.

[17] A. N. S. Praveen Seshadri. Generalized partial indexes. *ICDE*, pages 420–427, 1995.

[18] K. Schnaitter et al. COLT: Continuous On-Line Database Tuning. SIGMOD 2006.

[19] M. Stonebraker. The case for partial indexes. *SIGMOD Record*, 18(4):4–11, 1989.

[20] G. Valentin et al. DB2 Advisor: An Optimizer Smart Enough to Recommend Its Own Indexes. ICDE. 2000.

[21] D. C. Zilio et al. DB2 Design Advisor: Integrated Automatic Physical Database Design. VLDB 2004.