### ICDE 2018

# Adaptive Adaptive Indexing

Felix Martin Schuhknecht

Jens Dittrich

Laurent Linden

Big Data Analytics Group bigdata.uni-saarland.de

Saarland University

### 2007

### **Database Cracking**

Stratos Idreos CWI Amsterdam The Netherlands Stratos.ldreos@cwi.nl Martin L. Kersten CWI Amsterdam The Netherlands

Stefan Manegold CWI Amsterdam The Netherlands Martin.Kersten@cwi.nl Stefan.Manegold@cwi.nl

### ABSTRACT

ABSTRACT

Database indices provide a non-discriminative navigational infrastructure to localize tuples of interest. Their maintenance cost is taken during database updates. In this paper, we study the complementary approach, addressing index maintenance as part of query processing using continuous physical coroganization, i.e., crucking the database into manageable pieces. The motivation is that by automatically organizing data the way users request it, we can achieve fast access and the much desired self-organized behavior.

We present the first mature cracking architecture and report on our implementation of cracking in the context of a full fledged relational system. It led to a minor enhancement to its relational algebra kernel, such that cracking could be piggy-backed without incurring too much processing overhead. Furthermore, we illustrate the ripple effect of dynamic reorganization on the query plans derived by the SQL optimizer. The experiences and results obtained are indicative of a significant reduction in system complexity. We show that the resulting system is able to self-organize based on incoming requests with clear performance benefits. This behavior is visible even when the user focus is randomly shifting to different parts of the data.

### 1. INTRODUCTION

1. INTRODUCTION

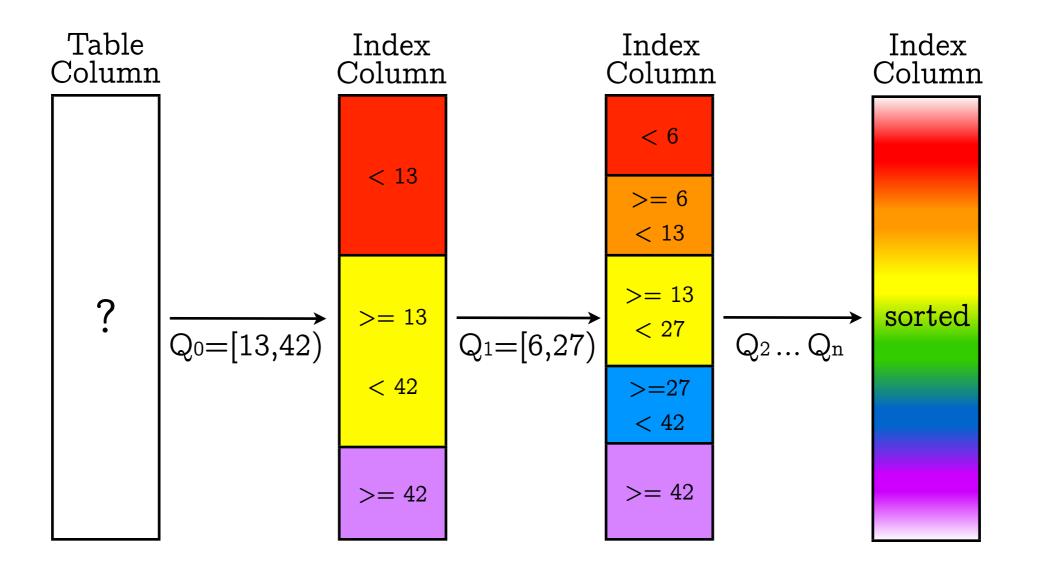
Nowadays, the challenge for database architecture design is not in achieving ultra high performance but to design systems that are simple and flexible. A database system should be able to handle huge sets of data and acf-organize according to the environment, e.g., the workload, available resources, etc. A nice discussion on such issues can be found in [6]. In addition, the trend towards distributed environments to speed up computation calls for new architecture designs. The same holds for multi-core CPU architectures that are starting to dominate the market and open new possibilities and challenges for data management. Some notable departures from the usual paths in database architecture design include [2, 3, 9, 14].

In this paper, we explore a radically new approach in database architecture, called database crucking. The approach is based on the hypothesis that index maintenance should be a byproduct of query processing, not of updates. Each query is interpreted not only as a request for a particular result set, but also as an advice to crack the physical database store into smaller pieces. Each piece is described by explored the control of the processing of the processing the control of the c

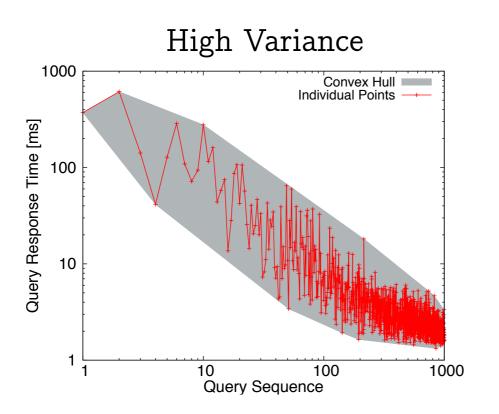
SQL micro-benchmarks, we assess the efficiency and effectiveness of the system at the operator level. Additionally, we perform experiments that use the complete observae stack, demonstrating that tracker-aware query optimizers can successfully generate query plans that deploy a perform exercising the performance optimizers of the system of the performance optimizers can successfully generate query plans that deploy and perform the work may copy, distribute, display, and perform the work, make derivative works to the author and CIDR 2007.

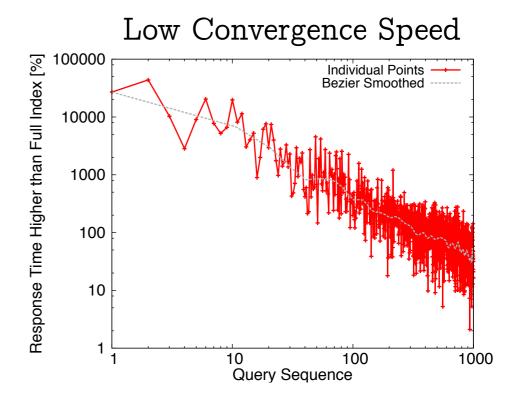
SQL micro-benchmarks, we assess the efficiency and efficiency

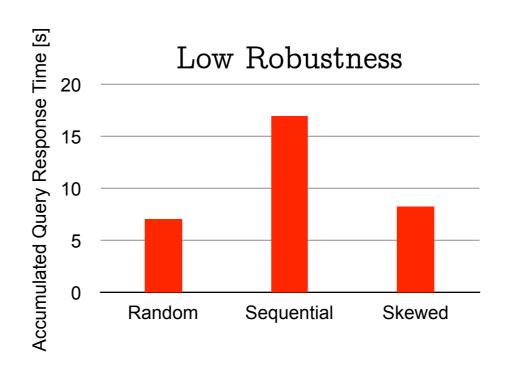
# Database Cracking / Standard Cracking



### Problems?







### Stochastic Database Cracking: Towards Robust Adaptive Indexing in Main-Memory Column-Stores'

Felix Halim Stratos Idreos Panagiotis Karras Roland H. C. Yap \*National University of Singapore {halim, ryap}@comp.nus.edu.sg †CWI, Amsterdam idreos@cwi.nl Rutgers University karras@business.rutgers

### ABSTRACT

Modern business applications and scientific databases call for in-herently dynamic data storage environments. Such environments are characterized by two challenging features: (a) they have lit-tle dide system time to devote on physical design, and (b) there is little, if any, a priori workload knowledge, while the query and data workload keeps changing dynamically. In such environments, traditional approaches to index building and maintenance cannot dy. Database cracking has been proposed as a solution that also on-the-fly physical data reorganization, as a collateral effect of lows on-the-fly physical data reorganization, as a collateral effect of query processing. Cracking aims to continuously and automatically adapt indexes to the workload at hand, without human intervention. Indexes are built incrementally, adaptively, and on dermad. Nevertheless, as we show, existing adaptive indexing methods fail to deliver workload-nobusness; they perform much better with random workloads than with others. This frailty derives from the inelasticity with which these approaches interpret each query as a hint on how data should be stored. Current eracking schemes blindly reorganize the data within each query's range, even if that results into successive expensive operations with minimal indexing benefit. In this paper, we introduce subchastic cracking, a significantly more resilient approach to adaptive indexing. Stochastic eracking also uses each query as a hint on how to roorganize data, but not

more resilient approach to adaptive indexing. Stochastic cracking also uses each query as a hint on how to reorganize data, but not blindly so; it gains resilience and avoids performance bottlenecks by deliberately applying certain arbitrary choices in its decisionof democracity applying certain aroundy choices in its excision-making. Thereby, we bring adaptive indexing forward to a ma-ture formulation that confers the workload-robustness previous ap-proaches lacked. Our extensive experimental study verifies that stochastic cracking maintains the desired properties of original da-tabase cracking while at the same time it performs well with diverse realistic workloads.

### 1. INTRODUCTION

Database research has set out to reexamine established assump-tions in order to meet the new challenges posed by big data, sci-entific databases, highly dynamic, distributed, and multi-core CPU

Permission to make digital or hair personal or classroom use is grant not made or distributed for profit or bear this notice and the full clinton republish, to post on servers or to re permission and/or a fee. Articles fi heir results at The 38th Internations August 27th - 31st 2012, Istanbul, J Proceedings of the VLDB Endowmen Copyright 2012 VLDB Endowmen

environments. One of the major challenges is to create simple-to-use and flexible database systems that have the ability self-organize according to the environment [7].

Physical Design. Good performance in database systems largely relies on proper tuning and physical design. Typically, all tuning choices happen up front, assuming sufficient workload knowledge and idle time. Workhoad knowledge is necessary in order to deter-mine the appropriate tuning actions, while idle time is required in order to perform those actions. Modern database systems rely on auto-tuning tools to carry out these steps, e.g., (8, 8, 13, 1, 28).

Dynamic Environments. However, in dynamic environments, workload knowledge and idle time are scarce resources. For ex-

Dynamic Environments. However, in dynamic environments, wowkind almowledge and life time are scarce resources. For example, in scientific databases new data arrives on a daily or even hourly basis, while query patterns follow an exploratory path as the scientists try to interpret the data and understand the patterns observed; there is no time and knowledge to analyze and prepare a different physical design every hour or even every day. Traditional indexing presents three fundamental weaknesses in such cases: (a) the workload may have changed by the time we finish tuning; (b) there may be no time to finish tuning properly; and (c) there is no indexing support during tuning.

Database Cracking. Recently, a new approach to the physical design problem was proposed, namely database cracking [14]. Cracking introduces the notion of continuous, incremental, partial and on demand adaptive indexing. Thereby, indexes are incrementally built and refined during query processing. Cracking was proposed in the context of modern column-stores and has been hitterion applied for boossting the performance of the select operator best opportuned to the select operator best particular or context of modern column-stores and has been hitterion applied for boossting the performance of the select operator erto applied for boosting the performance of the select operator [16], maintenance under updates [17], and arbitrary multi-attribute queries [18]. In addition, more recently these ideas have been ex-

queries [18], in adounts, more recently mese locas have ocen ex-tended to exploit a partition/merge-like logic [19], [11, 22]. Workload Robustness, Nevertheless, existing cracking schemes have not deeply questioned the particular way in which they in-terpret queries as a hint on how to organize the data store. They have adopted a simple interpretation, in which a select operator is taken to describe a range of the data that a discriminative cracker indicx should provide easy access to for future queries; the remain-der of the data remains non-indexed until a query expresses inter-

### **Database Cra**

Stratos Idreos CWI Amsterdam The Netherlands tos.ldreos@cwi.nl

to 1

diese provide a non-discriminative navigational
to to localize tuples of interest. Their maintestaken during database updates. In this payie complementary approach, addressing inance as part of query processing using continureorganization, i.e., crucking the database into
pieces. The motivation is that by automatically
ata the way users request it, we can achieve fast
he much desired self-organized behavior.
It he first mature cracking architecture and reimplementation of cracking in the context of a
stational system. It led to a minor enhancement
asi agebra kernel, such that cracking could be
i without incurring too much processing overwore, we illustrate the ripple effect of dynamic
so the query plans derived by the SQL optispeciences and results obtained are indicative of
reduction in system complexity. We show that eriences and results obtained are indicative of function in system complexity. We show that stem is able to self-organize based on incom-th clear performance benefits. This behavior when the user focus is randomly shifting to

Martin L. Kerste CWI Amsterdam The Netherlands Martin.Kersten@c

### Self-organizing Tuple Reconstruction in Column-stores

Stratos Idreos CWI Amsterdam The Netherlands

Martin L. Kersten CWI Amsterdam The Netherlands mk@cwi.nl

Stefan Manegold CWI Amsterdam The Netherlands manegold@cwi.nl

### ABSTRACT

ABSTRACT

Colums-stores gained popularity as a promising physical design alternative. Each attribute of a relation is physically stored as a separate column allowing queries to load only the required attributes. The overhead incurred is on-the-fly tuple reconstruction for multi-attribute queries. Each tuple reconstruction for soft of two columns based on tuple IDs, making it a significant cost component. The ultimate physical design is to have multiple presorted copies of each base table such that tuples are already appropriately organized in multiple different orders across the various columns. This requires the ability to predict the workload, idle time to prepare, and infrequent updates.

In this paper, we propose a novel design, partial side-weaps cracking, that minimizes the tuple reconstruction cost in a self-corranties was it. Satisves performance similar as

In this paper, the propose a novel design, partial state-uogs crucking, that minimizes the tuple reconstruction cost in a self-organizing way. It achieves performance similar to using presorted data, but without requiring the heavy initial presorting step itself. Instead, it handles dynamic, unpredictable workloads with no idle time and frequent up-dates. Auxiliary dynamic data structures, called crucker maps, provide a direct mapping between pairs of attributes used together in queries for tuple reconstruction. A map is continuously physically reorganized as an integral part of query evaluation, providing faster and reduced data access for future queries. To enable flexible and self-toganizing be-havior in storage-limited environments, maps are material-ized only partially as demanded by the workload. Each map is a collection of separate churks that are individually reor-ganized, dropped or recreated as needed. We implemented partial sideways cracking in an open-source column-store. A detailed experimental analysis demonstrates that it brings significant performance benefits for multi-attribute queries. Categories and Subject Descriptors: It 2, IDATABASE Categories and Subject Descriptors: H.2 [DATABASE MANAGEMENT]: Physical Design - Systems

General Terms: Algorithms, Performance, Design

Keywords: Database Cracking, Self-organization

### 1. INTRODUCTION

1. INTRODUCTION
A prime feature of column-stores is to provide improved performance over row-stores in the case that workloads require only a few attributes of white tables at a time. Each relation R is physically stored as a set of columns; one column for each attribute of R. This way, a query needs to load only the required attributes from each relevant relation. This happens at the expense of requiring explicit (partial) upple reconstruction in case multiple attributes are required.

tuple reconstruction in case multiple attributes are required. Each tuple reconstruction is a join between two columns based on tuple IDs/positions and becomes a significant cost component in column-stores especially for multi-attribute queries [2, 6, 10]. Whenever possible, position-based join-

component in countin-acores especially for multi-attribute queries [2, 6, 10]. Whenever possible, position-based join-matching and sequential data access are exploited. For each relation  $R_i$  in a query plan  $q_i$  a column-store needs to per-form at least  $N_i - 1$  tuple reconstruction operations for  $R_i$ within  $q_i$  given that  $N_i$  attributes of  $R_i$  participate in  $q_i$ . Column-stores perform tuple reconstruction in two ways [2]. With early tuple reconstruction, the required attributes are gized together as early as possible, i.e., while the columns are loaded, leveraging  $N_i$  any processing to evaluate the query. On the other hand, late tuple reconstruction exploits the column-store architecture to its maximum. During query processing, "seconstruction" merely refers to getting the at-tribute values of qualifying tuples from their base columns as late as possible, i.e., only once an attribute is required in the query plan. This approach allows the query engine to exploit CPU- and cache-optimized vector-like operator implementa-tions throughout the whole query evaluation.  $N_i$  ary tuples are formed only once the final result is delivered. Like most modern column-stores [12, 4, 15], we focus on

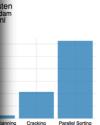
are formed only once the final result is delivered.

Like most modern column-stores [24, 4, 15], we focus on late reconstruction. Comparing early and late reconstruction, the educative analysis in [2] observes that the latter incurs the overhead of reconstructing a column more than once, in case it occurs more than once in query. Furthermore, exploiting sequential access patterns during reconstruction is not always possible since many operators [joins, group by, order by etc.] are not tuple order-preserving. The ultimate access pattern is to have multiple copies for each relation R, such that each copy is presorted on an other attribute in R. All tuple reconstructions of R attributes initiated by a restriction on an attribute A can be performed using the copy that is sorted on A. This way, the tuple

Permission to make digital or hard copies of all or part of this work for using the copy that is sorted on A. This way, the tuple

### Poor Man's Sort!

Stratos Idreos Harvard University @seas.harvard.edu



Costs of Database Operations

of Cracking is fully sorted data, its costs are Cracking is fully sorted data, its costs are so of fully sorting the data. With recent harallel) sorting algorithms [7], however, singly unattractive. To illustrate this, Figmparison of the respective operations on a 4-Core Sandy Bridge CPU. he-shelf (Parallel) Mer the-shelf (Parallel) Mergesort implemen-more expensive than a (quasi I/O bound) y three times as expensive as MonetDB's ing [19]. Even though both Scanning and read and write the same amount of data, costs. The performance difference must, costs. Tracking is, unlike Scan-

nted with the underlying hardware in

thesis, we make the following contributions t an in-depth study of the contributing performance the "classic" Cracking implementation.

the findings, we develop a number of optimizaased on "standard" techniques like predication, vec-on and manually implemented data parallelism using

two different parallel algorithms that exploit thread lism to make use of multiple CPU cores.

ly evaluate all developed algorithms on a number ent systems ranging from low-end desktop machines

NU libstdc++ Version 4.8.2

### Merging What's Cracked, Cracking What's Merged: Adaptive Indexing in Main-Memory Column-Stores

Stratos Idreos† Stefan Manegold†

†CWI, Amsterdam {stratos.idreos, stefan.manegold}@cwi.nl

Harumi Kuno\* Goetz Graefe\*

Adaptive indexing is characterized by the partial creation and refinement of the index as side effects of query execution. Dynamic or shifting workloads may benefit from preliminary index structures focused on the columns and specific key ranges actually queried—without incurring the cost of full index construction. The costs and benefits of adaptive indexing techniques should therefore be compared in terms of iritialization costs, the overhead imposed in the compared in terms of iritialization costs, the overhead imposed in the compared in terms of iritialization costs, the overhead imposed in the compared in terms of iritialization costs, the overhead imposed in the compared in terms of iritialization costs, the overhead imposed in the compared in terms of iritialization costs, the overhead imposed in the compared in upon queries, and the rate at which the index converges to a state that is fully-refined for a particular workload component.

Based on an examination of database cracking and adaptive merg-ing, which are two techniques for adaptive indexing, we seek a hybrid technique that has a low initialization cost and also converges rapidly. We find the strengths and weaknesses of database cracking and adaptive merging complementary. One has a relatively high initialization cost but converges relatively. The other has a low initialization cost but converges relatively alonly. We analyze the sources of their respective strengths and explore the space of hybrid electhiques. We have designed and implemented a famility of hybrid algorithms in the context of a column-store database system. Our experiments compare their behavior against database system. Our experiments compare their behavior against database cracking and adaptive merging, as well as against both traditional full index lookup and scan of unordered data. We show that the new hybrids significantly improve over past methods while at least two of the hybrids some very close to the "ideal performance" in terms of both overhead per query and convergence to a final state. hybrid technique that has a low initialization cost and also con-

### 1. INTRODUCTION

1. INTRODUCTION
Contemporary index selection tools rely on monitoring database requests and their execution plans, occasionally invoking creation or removal of indexes on tables and views. 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 reaction efforts, and easternly index reaction imposes an additional. ion efforts, and eventual index creation imposes an additional

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profits or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to permission and/or a relative present page and the copies bear this notice and the full citation on the first page. To copy otherwise, to permission and/or a fee. Articles from this volume were invited to present their results at The 37th International Conference on Very Large Data Bases, Arquist 20th - September 3rd 2011, Seattle, Washington, Proceedings of the VLDB Endowment 120-8-8079/1106... \$10.00.

\*HP Labs, Palo Alto {harumi.kuno, goetz.graefe}@hp.com



### Figure 1: Adaptive Indexing Research Space

Figure 1: Adaptive Indexing Research Space.

In that interferse 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.

Our goal is 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. As proposed in [5], we use two measures 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 beauty benefits from the index structure without incurring any overhead. We focus particularly on the first query because it captures the worst-case costs and benefits of adaptive indexing; if that portion of data is never queried again, then any overhead above and beyond the cost of a

queried again, then any overhead above and beyond the cost of a scan is wasted effort. Recent work has proposed two distinct approaches: database cracking [10, 11, 12] and daptive merging [6, 7]. The more of-ten a key range is queried, he more its representation is optimized. Columns that are not queried are not indended, 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. In order to evaluate database cracking and adaptive merging, we have implemented both approaches in a modern column-store database system, and find the strengths and weaknesses of the two approaches complementary.

As shown in Figure 1, adaptive merging has a relatively high initialization cost but converges rapidly, while database cracking enjoys a low initialization cost to converge seatively slowly. The green box in Figure 1 thus defines the research space for adaptive indexing with database cracking and adaptive merging occupying the borders of this space. We recognize the opportunity for an ideal hybrid adaptive indexing technique, marked with a star in the figure, that incurs a low initialization cost yet also converges quickly Recent work has proposed two distinct approaches: database

### Self-selecting, self-tuning, incrementally optimized indexes

Goetz Graefe Hewlett-Packard Laboratories 1501 Page Mill Road Palo Alto, CA 94304

### Abstract

In a relational data warehouse with many tables, the in a retational data warenouse with many taoles, the number of possible and promising indexes exceeds human comprehension and requires automatic index tuning. While monitoring and reactive index tuning have been proposed, adaptive indexing focuses on adapting the physical data-base layout for and by actual queries.

"Database cracking" is one such technique. Only if and when a column is used in query predicates, an index for the column is created; and only if and when a key range is que-ried, the index is optimized for this key range. The effect is akin to a sort that is adaptive and incremental. This sort is. and to a soft that is adaptive and interferental. Into soft is, however, very inefficient, particularly when applied on block-access devices. In contrast, traditional index creation sorts data with an efficient merge sort optimized for block-access devices, but it is neither adaptive nor incremental.

We propose adaptive merging, an adaptive, incremental, and efficient technique for index creation. Index optimization focuses on key ranges used in actual queries. The resulting index adapts more quickly to new data and to new query patterns than database cracking. Sort efficiency is comparable to that of traditional B-tree creation. None theless, the new technique promises better query performance than database cracking, both in memory and on block-

### Categories and subject descriptors

### Keywords Database index, adaptive, autonomic, query execution.

1 Introduction

### In a relational data warehouse with a hundred tables and a thousand columns, billions of indexes are possible, in particular if partial indexes, indexes on computed columns,

Permission to make digital or hard copies of all or part of this Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

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

Copyright 2010 ACM 978-1-60558-945-9/10.0003 ...\$10.00.

Hewlett-Packard Laboratories 1501 Page Mill Road Palo Alto, CA 94304

and materialized views with their indexes are considered. Thus, index selection is a central, classic, and very hard rinds, muck selection is a ceitain, trassic, and very many problem in physical database design. Too few or the wrong indexes force many queries to scan large parts of the data-base; too many indexes force high update costs. Unpredict-able ad-hoc queries exacerbate the problem.

One approach is to focus on enabling very fast scans, e.g., using shared scans and columnar storage formats, an approach suitable to high-bandwidth high-latency devices such as traditional disk drives and disk arrays. Low-latency database storage such as flash memory will likely reenergize research into index-based query processing.

Another approach is to tune indexes in response to the actual workload. Contemporary index selection tools rely on monitoring database requests and their execution plans, occasionally invoking creation or removal of indexes on occasionary invoking creation or removal or indexes on tables and views. Such tools tend to suffer from 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, so data access during the interval port during this interval, so data access during the interval is wasted with respect to index creation, 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. For example, recent business transactions are queried more often than those years ago, extreme price fluctuations are more interesting than stable prices, etc. Even where it is possible to limit an index, e.g., using a partial index or a materialized view, it is often difficult or impossible to predict the key ranges to focus on.

Database cracking [IKM 07a, KM 05] has nioneered Database cracking [IKM 07a, KM 05] has pioneered focused, incremental, automatic optimization of the representation of a data collection – the more often a key range is queried, the more its representation is optimized. This optimization occurs entirely automatically, as a side effect of queries over key ranges not yet fully optimized.



For example, after the column store illustrated in Figure 1 has been queried with range boundary values c, g, m, s, and u, all key values below c have been assigned to

5/20

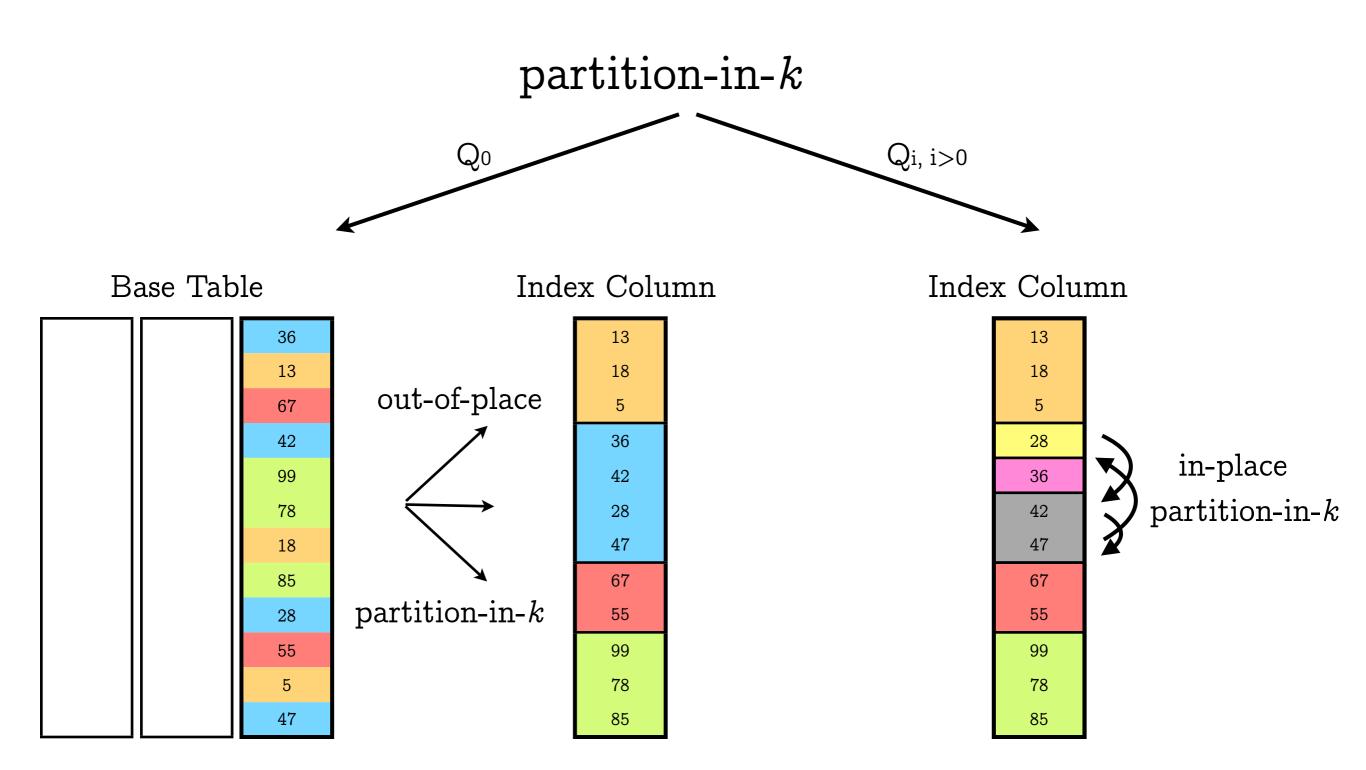
All-in-one?

An Adaptive Adaptive Index?

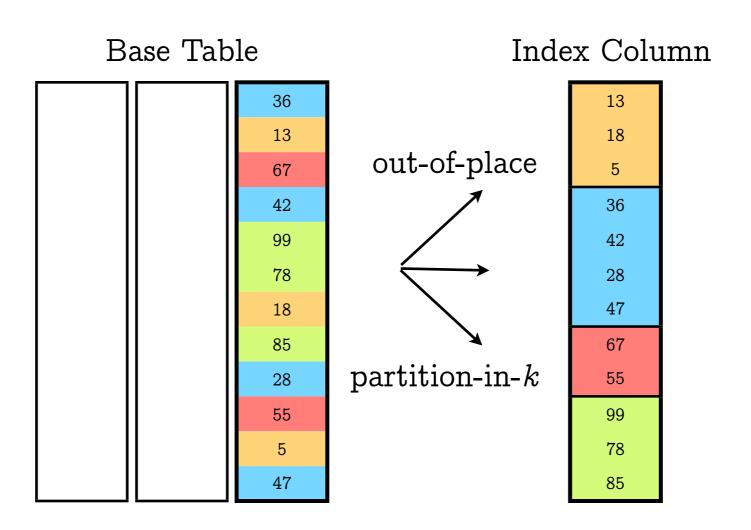
### Design rules:

- 1. Generalize way of refinement
- 2. Adapt refinement effort
- 3. Awareness of key distributions

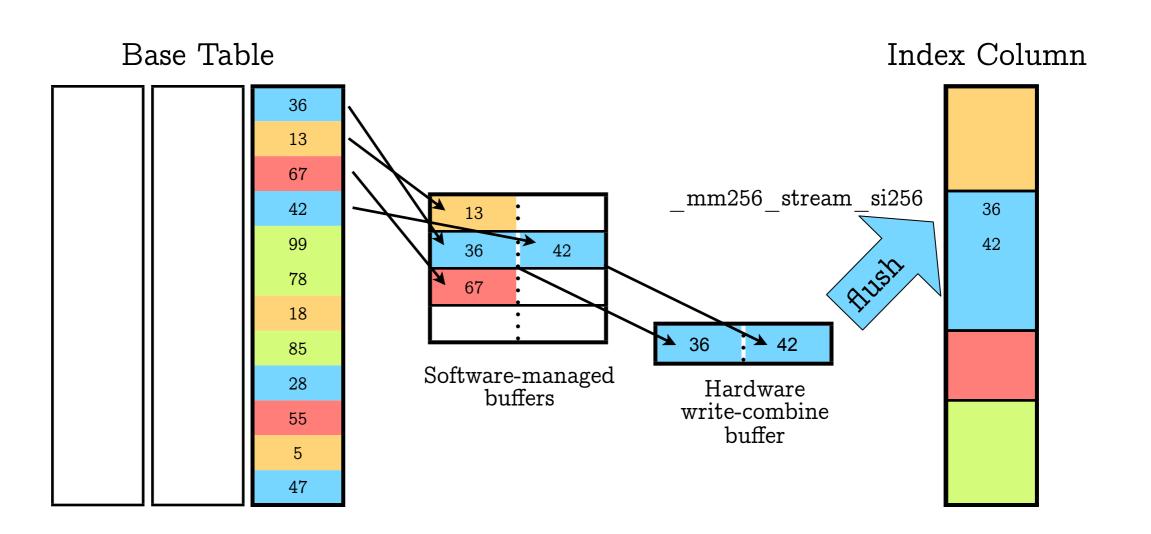
## 1. Generalize way of refinement:



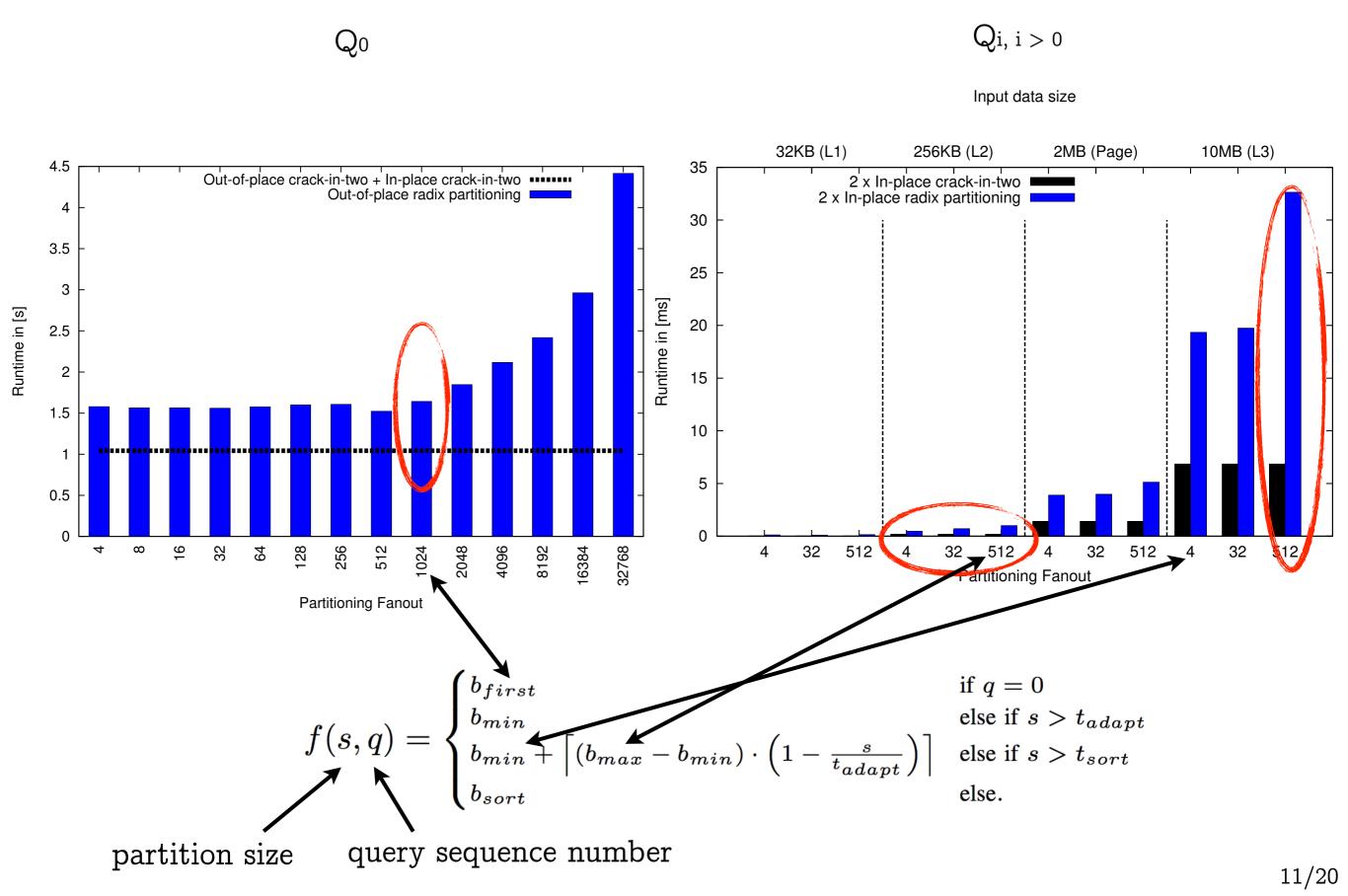
# 1. Generalize way of refinement:



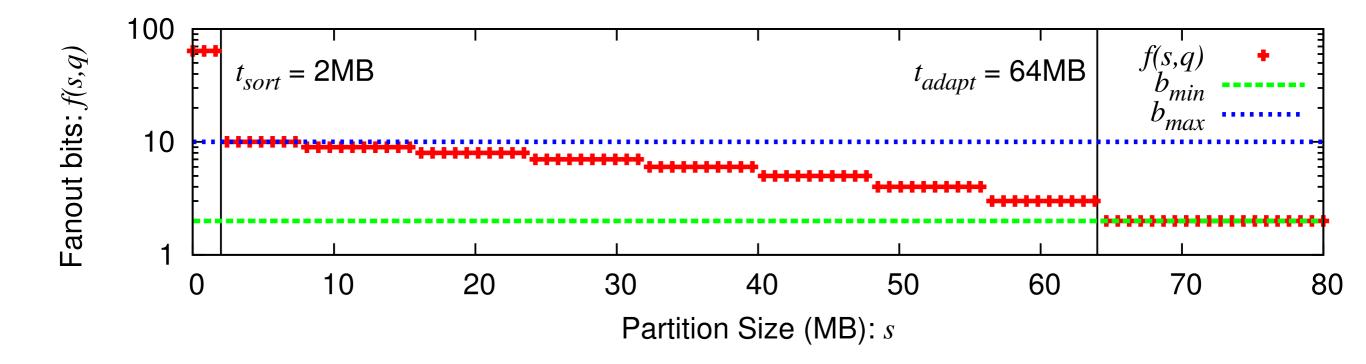
# 1. Generalize way of refinement:



### 2. Adapt refinement effort

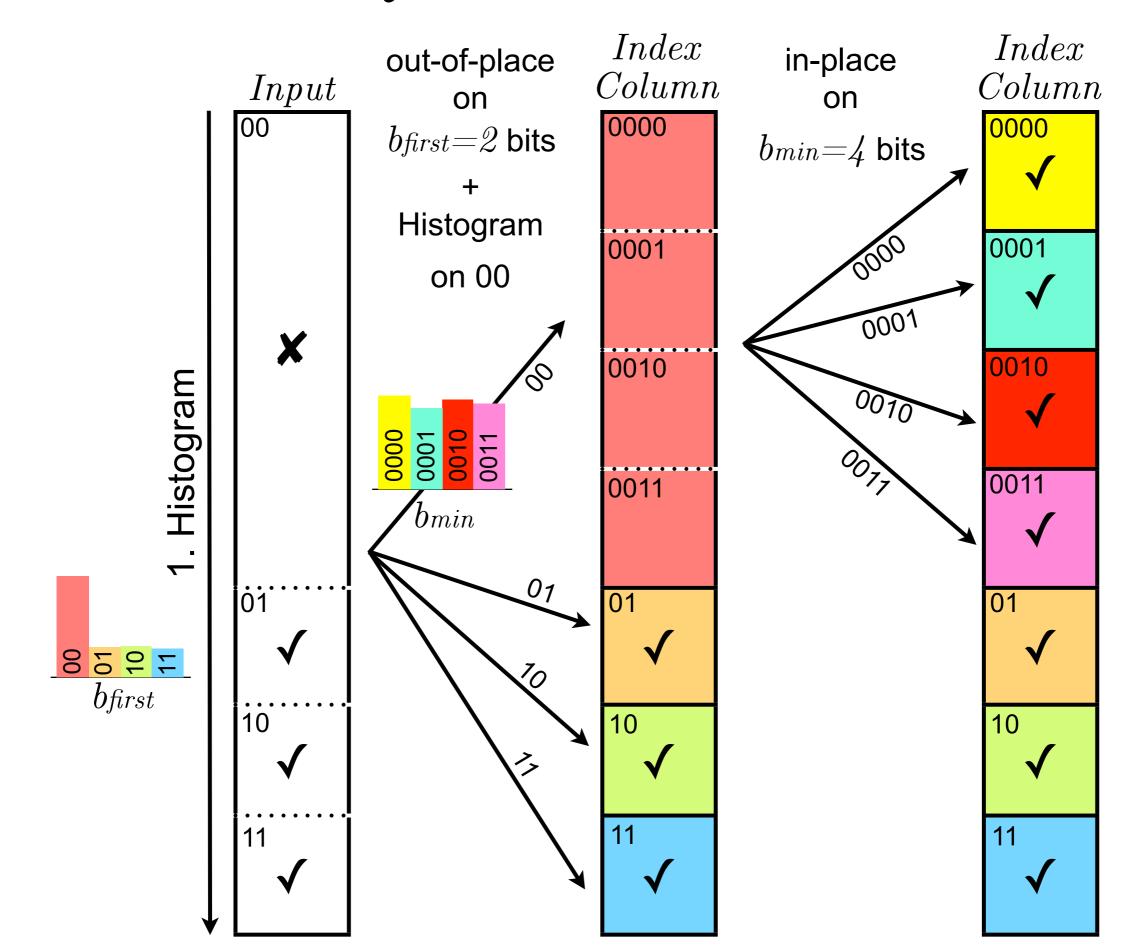


## 2. Adapt refinement effort

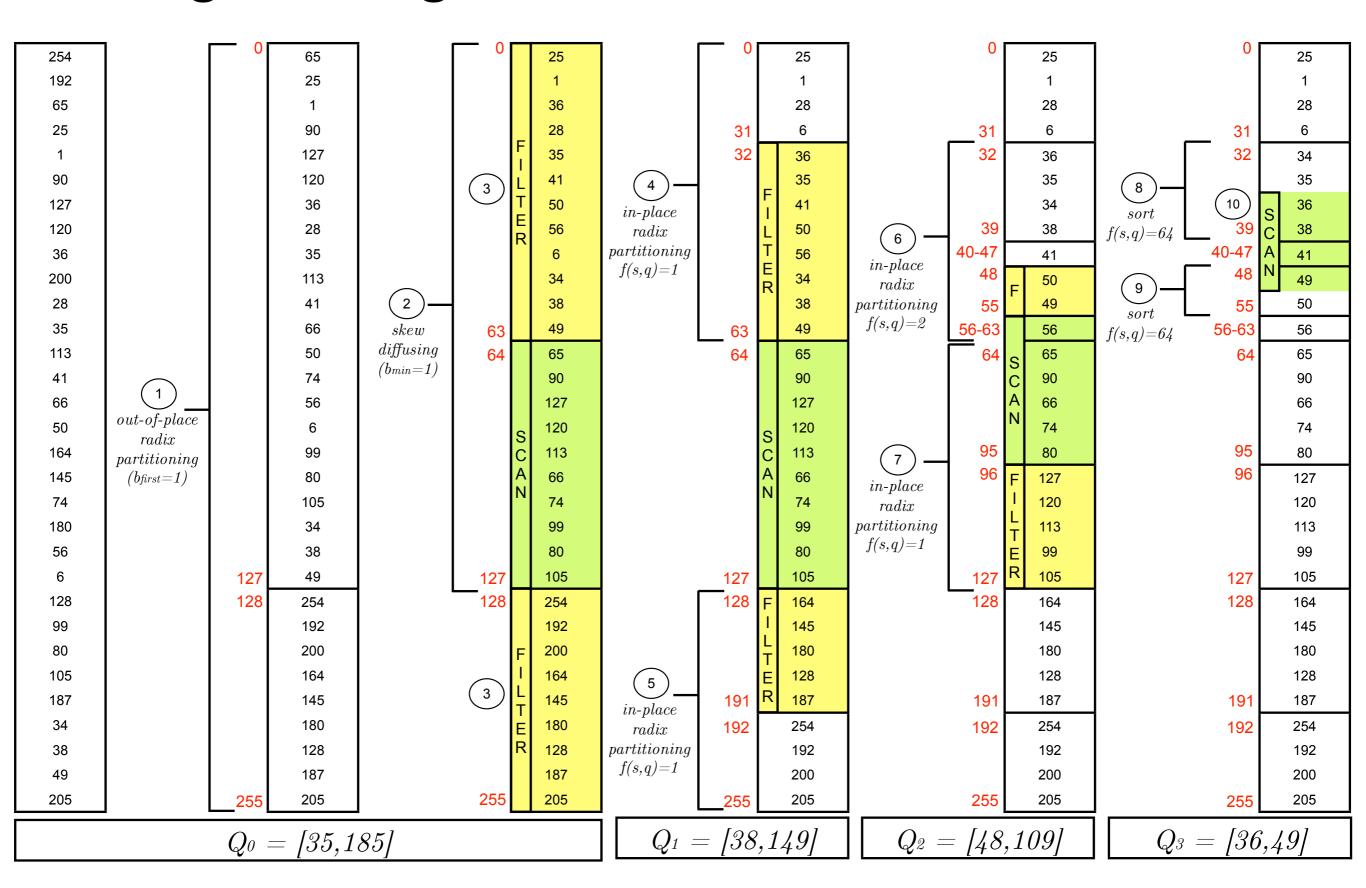


$$f(s,q) = \begin{cases} b_{first} & \text{if } q = 0\\ b_{min} & \text{else if } s > t_{adapt}\\ b_{sort} & \text{else if } s > t_{sort} \end{cases}$$

# 3. Awareness of key distributions: skew?

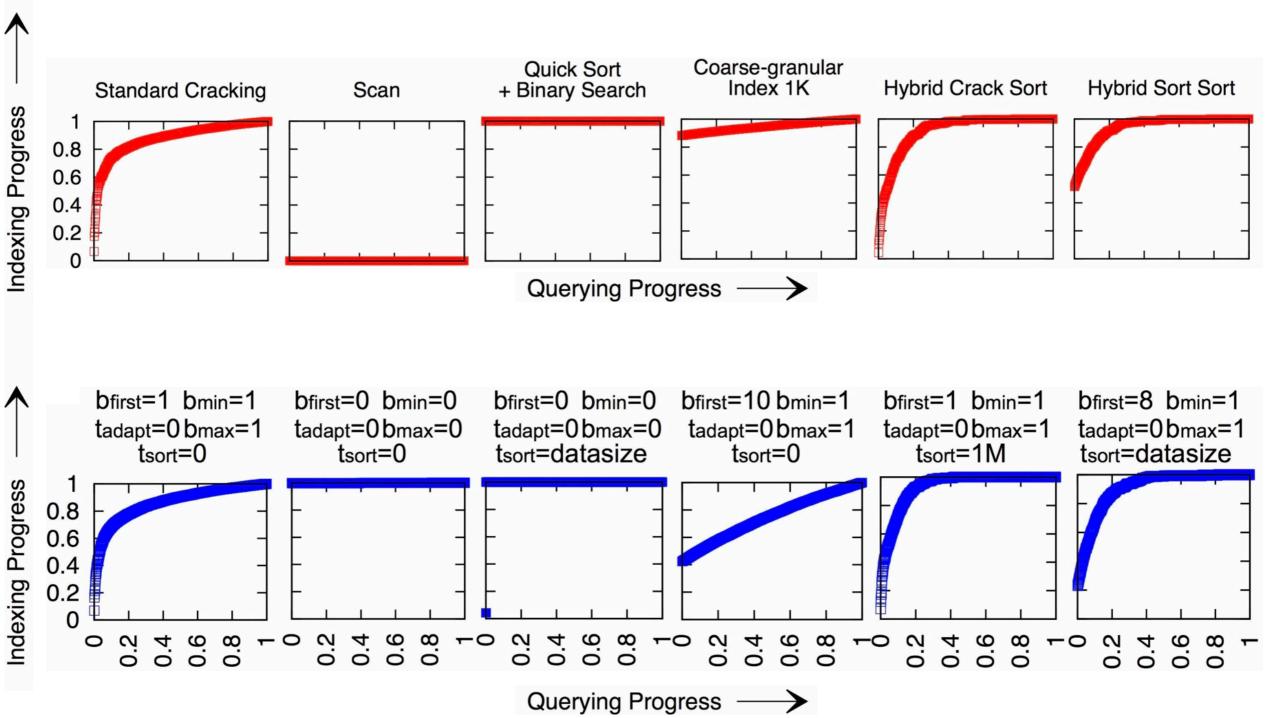


# Putting it all together

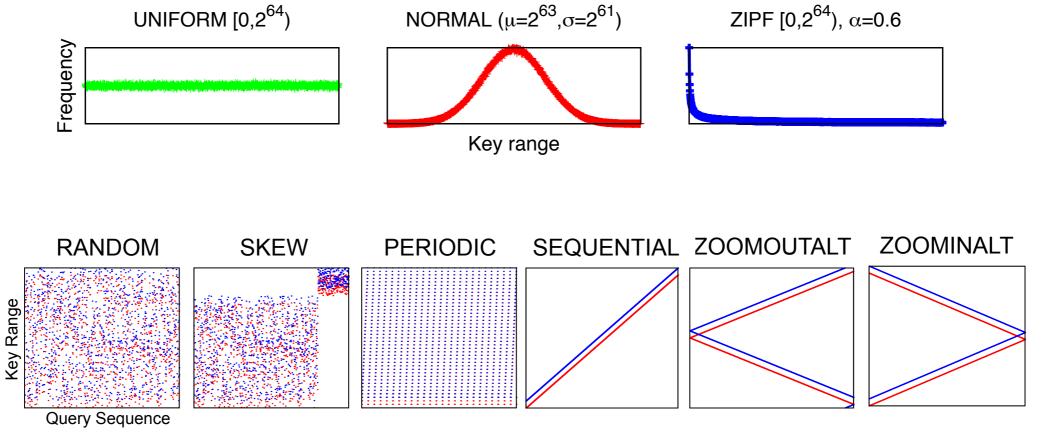


### **Emulation**

[Felix Martin Schuhknecht, Alekh Jindal, Jens Dittrich: The Uncracked Pieces in Database Cracking, PVLDB Vol. 7, No. 2]

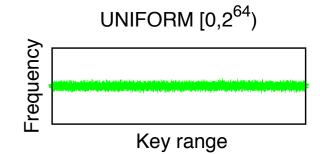


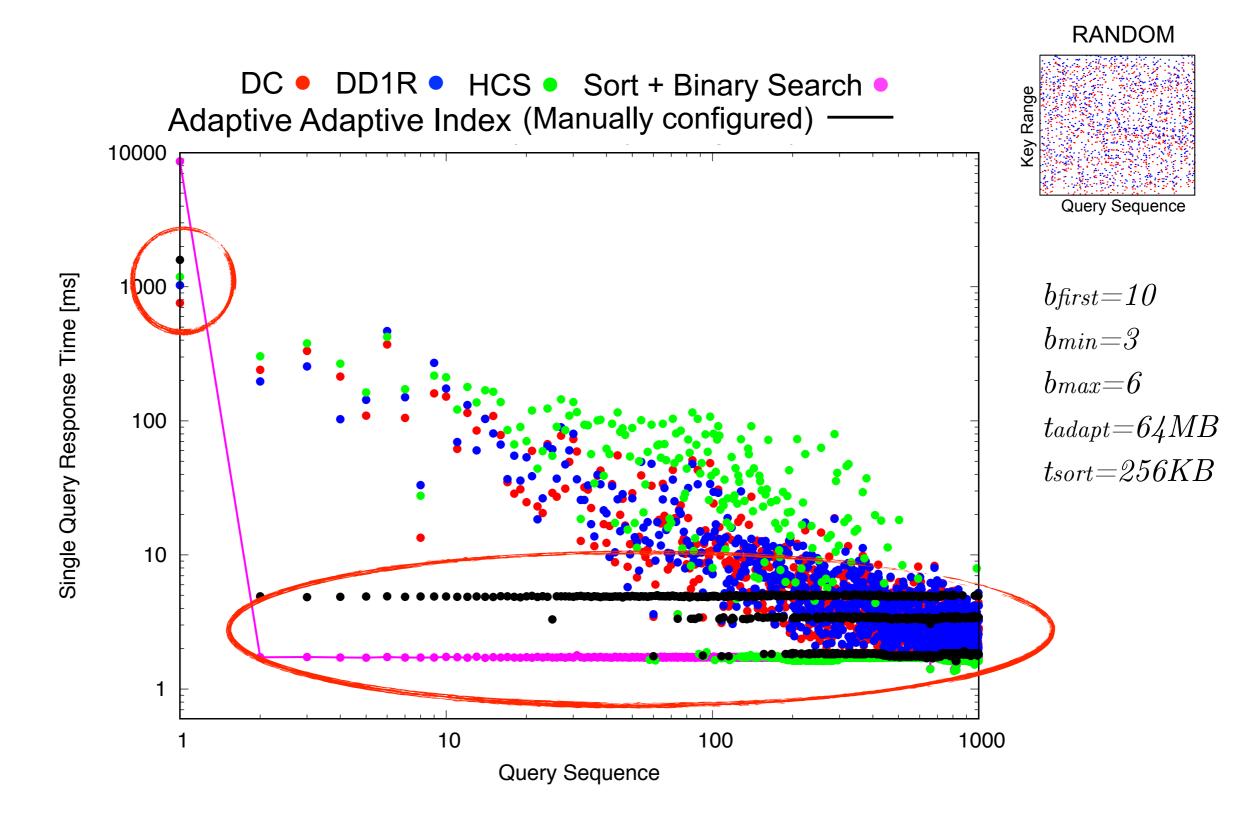
### Test Setup



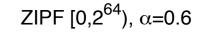
[Felix Halim, Stratos Idreos, Panagiotis Karras, Roland H. C. Yap: Stochastic Database Cracking: Towards Robust Adaptive Indexing in Main-Memory Column-Stores, PVLDB Vol. 5, No. 6]

### Individual Query Response Times



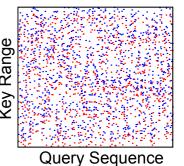


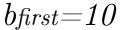
### Individual Query Response Times



Key range



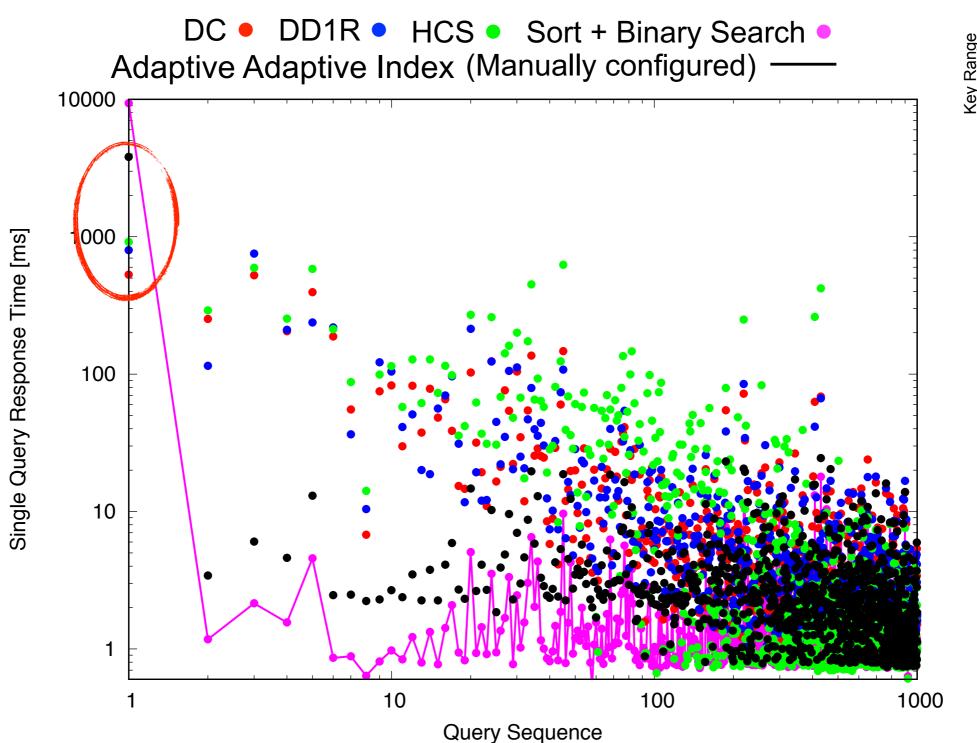




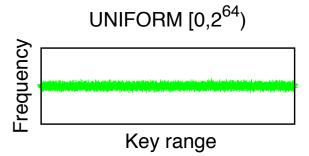
$$bmin=3$$

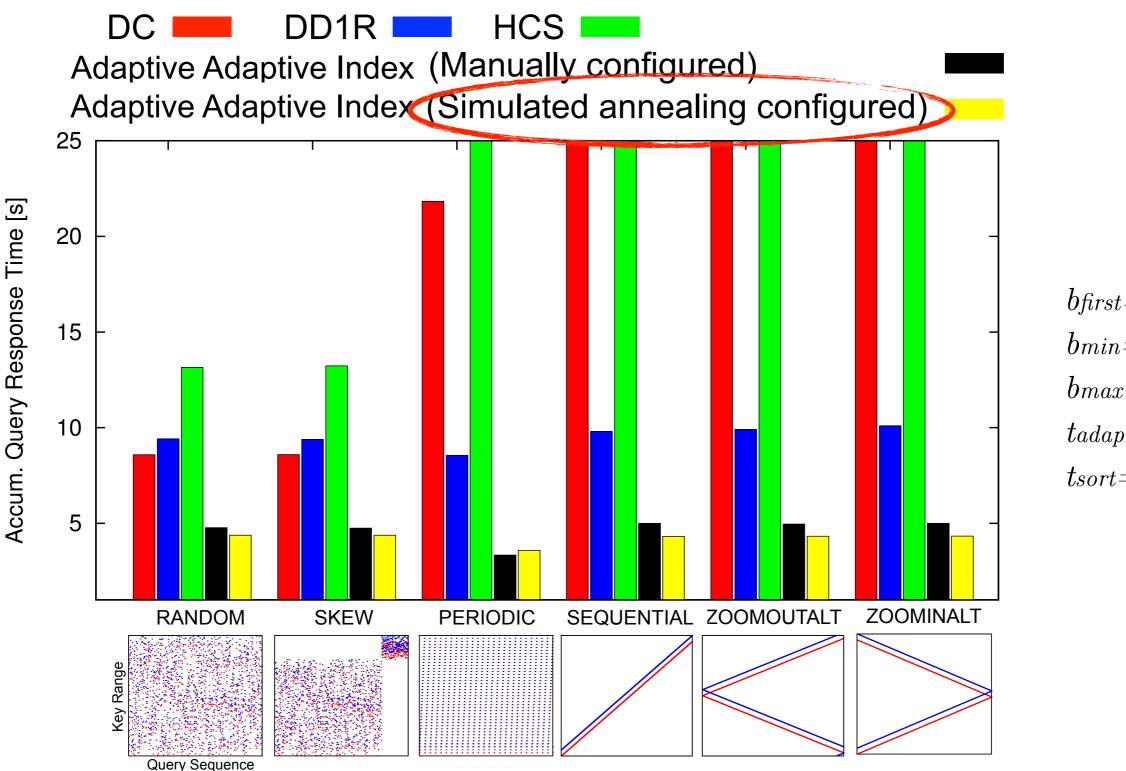
$$b_{max}=6$$

$$\mathit{tsort} = 256KB$$



## Accumulated Query Response Times





bfirst=10 bmin=3 bmax=6 tadapt=64MB tsort=256KB

### Stochastic Database Cracking: Towards Robust Adaptive Indexing in Main-Memory Column-Stores'

Felix Halim Stratos Idreos Panagiotis Karras Roland H. C. Yap Rutgers University karras@business.rutgers

### ABSTRACT

Modern business applications and scientific databases call for in-herently dynamic data storage environments. Such environments are characterized by two challenging features: (a) they have lit-tle dide system time to devote on physical design, and (b) there is little, if any, a priori workload knowledge, while the query and data workload keeps changing dynamically. In such environments, traditional approaches to index building and maintenance cannot dy. Database cracking has been proposed as a solution that also on-the-fly physical data reorganization, as a collateral effect of lows on-the-fly physical data reorganization, as a collateral effect of query processing. Cracking aims to continuously and automatically adapt indexes to the workload at hand, without human intervention. Indexes are built incrementally, adaptively, and on dermad. Nevertheless, as we show, existing adaptive indexing methods fail to deliver workload-nobusness; they perform much better with random workloads than with others. This frailty derives from the inelasticity with which these approaches interpret each query as a hint on how data should be stored. Current eracking schemes blindly reorganize the data within each query's range, even if that results into successive expensive operations with minimal indexing benefit. In this paper, we introduce subchastic cracking, a significantly more resilient approach to adaptive indexing. Stochastic eracking also uses each query as a hint on how to roorganize data, but not

more resilient approach to adaptive indexing. Stochastic cracking also uses each query as a hint on how to reorganize data, but not blindly so; it gains resilience and avoids performance bottlenecks by deliberately applying certain arbitrary choices in its decisionby demoratory applying tertain aroutary crooses in its occession-making. Thereby, we bring adaptive indexing forward to a ma-ture formulation that confers the workload-robustness previous ap-proaches lacked. Our extensive experimental study verifies that stochastic cracking maintains the desired properties of original da-tabase cracking while at the same time it performs well with diverse realistic workloads.

### 1. INTRODUCTION

Database research has set out to reexamine established assump-tions in order to meet the new challenges posed by big data, sci-entific databases, highly dynamic, distributed, and multi-core CPU

Permission to make digital or hair personal or classroom use is grant not made or distributed for profit or bear this notice and the full clinton republish, to post on servers or to re permission and/or a fee. Articles fi heir results at The 38th Internations August 27th - 31st 2012, Istanbul, J Proceedings of the VLDB Endowmen Copyright 2012 VLDB Endowmen

environments. One of the major challenges is to create simple-to-use and flexible database systems that have the ability self-organize according to the environment [7].

Physical Design. Good performance in database systems largely relies on proper nating and physical design. Typically, all tuning choices happen up front, assuring sufficient workload knowledge and side time. Workhoad knowledge is necessary in order to deter-mine the appropriate tuning actions, while idle time is required in order to perform those actions. Modern database systems rely on auto-tuning tools to carry out these steps, e.g., 6, 8, 13, 1, 28. Dynamic Environments. However, in dynamic environments, workload knowledge and life time are searce resources. For ex-

Dynamic Environments. However, in dynamic environments, workload knowledge and life time are scarce resources. For example, in scientific databases new data arrives on a daily or even hourly basis, while query patterns follow an exploratory path as the scientists try to interpret the data and understand the patterns observed; there is no time and knowledge to analyze and prepare a different physical design every hour or even every day. Traditional indexing presents three fundamental weaknesses in such cases: (a) the workload may have changed by the time we finish tuning; (b) there may be not time to finish tuning properly; and (c) there is no indexing support during tuning.

Database Cracking. Recently, a new approach to the physical design problem was proposed, namely database cracking [14]. Cracking introduces the notion of continuous, incremental, partial and on demand adaptive indexing. Thereby, indexes are incrementally built and refined during query processing. Cracking was proposed in the context of modern column-stores and has been hitterion applied for boossting the performance of the select operator best opportuned to the select operator best operators and the select operator best operator best operators and the select operator best operators. erto applied for boosting the performance of the select operator [16], maintenance under updates [17], and arbitrary multi-attribute queries [18]. In addition, more recently these ideas have been ex-

queries [18], in adounts, more recently trace locas nate obeen ex-tended to exploit a partition/merge-like logic [19, 11, 12]. Workload Robustness. Nevertheless, existing cracking schemes have not deeply questioned the particular way in which they in-terpret queries as a hint on how to organize the data store. They have adopted a simple interpretation, in which a select operator is taken to describe a range of the data that a discriminative cracker indicx should provide easy access to for future queries; the remain-der of the data remains non-indexed until a query expresses inter-

### Merging What's Cracked, Cracking W Adaptive Indexing in Main-Memory C

Stratos Idreos† Stefan Manegold† Harumi Kuno †CWI, Amsterdam {stratos.idreos, stefan.manegold}@cwi.nl

Adaptive indexing is characterized by the partial creation and re-finement of the index as side effects of query execution. Dynamic or shifting workloads may benefit from preliminary index struc-tures focused on the columns and specific key ranges actually queried— without incurring the cost of full index construction. The costs and benefits of adaptive indexing techniques should therefore be compared in terms of initialization costs, the overhead imposed upon queries, and the rate at which the index converges to a state that is fully-refined for a particular workload component.

Based on an examination of database cracking and adaptive merg-ing, which are two techniques for adaptive indexing, we seek a hybrid technique that has a low initialization cost and also converges rapidly. We find the strengths and weaknesses of database cracking and adaptive merging complementary. One has a relatively high initialization cost but converges relatively. The other has a low initialization cost but converges relatively alonly. We analyze the sources of their respective strengths and explore the space of hybrid electhiques. We have designed and implemented a famility of hybrid algorithms in the context of a column-store database system. Our experiments compare their behavior against database system. Our experiments compare their behavior against database cracking and adaptive merging, as well as against both traditional full index lookup and scan of unordered data. We show that the new hybrids significantly improve over past methods while at least two of the hybrids some very close to the "ideal performance" in terms of both overhead per query and convergence to a final state. hybrid technique that has a low initialization cost and also con-

### 1. INTRODUCTION

1. INTRODUCTION
Contemporary index selection tools rely on monitoring database requests and their execution plans, occasionally invoking creation or removal of indexes on tables and views. 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 accessed during the monitoring interval neither benefits from nor aids index reaction efforts, and earnthal index creation imposes an additional. ion efforts, and eventual index creation imposes an additional

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profits or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to permission and/or a relative present page and the copies bear this notice and the full citation on the first page. To copy otherwise, to permission and/or a fee. Articles from this volume were invited to present their results at The 37th International Conference on Very Large Data Bases, Arquist 20th - September 3rd 2011, Seattle, Washington, Proceedings of the VLDB Endowment 120-8-8079/1106... \$10.00.

Figure 1: Ad which high falls. After thes total and interferes will tonal indexes on table are needed often and sr Our goal is to enable i.e., index creation and cution, with the implicit ranges truly queried are measures to characterial adapts index structures with the interference of the information which key depends and reasonable initialization cost incurred by the first query and (2) the number of united the interference of the information of the interference of the information which key depends on the information which key demands of the information which key demands of the information which key demands on the informatio queried again, then any overhead above and beyond the cost of a scan is wasted effort.

Recent work has proposed two distinct approaches: database cracking [10, 11, 12] and daptive merging [6, 7]. The more of-ten a key range is queried, he more its representation is optimized. Columns that are not queried are not indended, 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. In order to evaluate database cracking and adaptive merging, we have implemented both approaches in a modern column-store database system, and find the strengths and weaknesses of the two approaches complementary.

As shown in Figure 1, adaptive merging has a relatively high initialization cost but converges rapidly, while database cracking enjoys a low initialization cost to converge seatively slowly. The green box in Figure 1 thus defines the research space for adaptive indexing with database cracking and adaptive merging occupying the borders of this space. We recognize the opportunity for an ideal hybrid adaptive indexing technique, marked with a star in the figure, that incurs a low initialization cost yet also converges quickly Recent work has proposed two distinct approaches: database

### Self-organizing Tuple Reconstruction in Column-stores

Martin L. Kersten CWI Amsterdam The Netherlands mk@cwi.nl

Stefan Manegold CWI Amsterdam The Netherlands manegold@cwi.nl

### ABSTRACT

ABSTRACI
Column-stores gained popularity as a promising physical design alternative. Each attribute of a relation is physically stored as a separate column allowing queries to load only the required attributes. The overhead incurred is on-the-fly tuple reconstruction for multi-attribute queries. Each tuple reconstruction is a join of two columns based on tuple IDs, making it a significant cost component. The ultimate physical design is to have multiple presorted copies of each base table such that tuples are already appropriately organized in multiple different orders across the various columns. This requires the ability to predict the workload, idle time to prepare, and infrequent updates.

In this paner, we propose a novel design, pertial side—

Adaptive Adaptive Indexing

Felix Martin Schuhknecht<sup>1</sup>, Jens Dittrich<sup>2</sup>, Laurent Linden<sup>3</sup> Saarland Informatics Campus Saarland University, Germany felix.schuhknecht@infosys.uni-saarland.de <sup>2</sup> jens.dittrich@infosys.uni-saarland.de

3 laurent.linden@gmx.net

Databa

adaptive indexes solely adapt the indexedness of the data to th workload. However, we will learn that so far we have overlooke a second higher level of adaptivity, namely the one of the indexin algorithm itself. We will coin this second level of adaptivit

meta-adaptivity.

Based on a careful experimental analysis, we will develop an adaptive index, which realizes meta-adaptivity by (1) generalizing the way reorganization is performed, (2) reacting to the evolving

the way reorganization is performed, (2) reacting to the evolving indexedness and varying reorganization effort, and (3) defusing skewed distributions in the input data. As we will demonstrate, this allows us to emulate the characteristics of a large set of specialized adaptive indexing algorithms. In an extensive experimental study we will show that our meta-adaptive index is extremely fit in a variety of environments and outperforms a large amount of specialized adaptive indexes under various query access patterns and key distributions.

I. INTRODUCTION

### 1. INTRODUCTION

1. INTRODUCTION
A prime feature of column-stores is to provide improved performance over row-stores in the case that workloads require only a few attributes of white tables at a time. Each relation R is physically stored as a set of columns; one column for each attribute of R. This way, a query needs to load only the required attributes from each relevant relation. This happens at the expense of requiring explicit (partial) upple reconstruction in case multiple attributes are required.

Each tuple reconstruction is a join between two colur based on tuple IDs/positions and becomes a significant of component in column-stores especially for multi-attribut queries [2, 6, 10]. Whenever possible, position-based join

matching and sequential data access are exploited. For each  $R_i$  in a query plan q, a column-store needs to per-R, in a query plan q, a column-store needs to per-seast N, -1 tuple reconstruction operations for R, given that N, attributes of R, participate in q, unstores perform tuple reconstruction in two ways [2], fy tuple reconstruction, the required attributes are gether as early as possible, i.e., while the columns d, leveraging N-ary processing to evaluate the query-cother hand, late tuple reconstruction exploits the tore architecture to its maximum. During query ig, "reconstruction" merely refers to getting the at-sulates of qualifying tuples from their base columns as sosible, i.e., only once an attribute is required in the sn. This approach allows the query engine to exploit d cache-optimized vector-like operator implementarins approach anows the query engine to exponent ache-optimized vector-like operator implementa-ghout the whole query evaluation. N-ary tuples only once the final result is delivered.

st only once the final result is delivered.

nost modern column-stores [22, 4, 15], we focus on natruction. Comparing early and late reconstruction and the construction and the reconstructing a column more than case it occurs more than once in a query. Further-ploiting sequential access patterns during reconsist not always possible since many operators (joins, order by etc.) are not tuple order-preserving, itimate access pattern is to have multiple copies for in R, such that each copy is presorted on an other in R. All tuple reconstructions of R attributes by a restriction on an attribute A can be performed a copy that is sorted on A. This way, the tuple

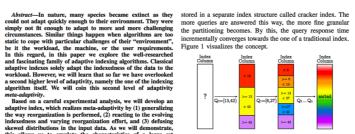


Fig. 1: Concept of database cracking reorganizing for multiple queries and converging towards a sorted state.

If we inspect the literature [4], [5], [6], [7], [8], [9], [10] proposing variations of the described principle, we see that these algorithms mostly focus on reducing a *single* issue at a time. For instance, hybrid cracking [5] tries to im-An overwhelming amount of adaptive indexing algorithms prove the convergence speed towards a full index. Stochastic exists today. In our recent studies [1], [2], we analyzed 8 pactures are speed towards a full index. Stochastic cracking [4] instead focuses on improving the robustness on pers including 18 different techniques on this type of indexing. sequential query workloads. Thus, to equip a system with The reason for the necessity of such a large number of adaptive indexing, it actually has to be extended with numer-methods is that adaptivity, while offering many nice properties, ous different implementations that must be switched depending introduces a surprising amount of unpleasant problems [1], on the needs of the user and the current workload.

[2] as well. For instance, as the investigation of these works showed, adaptive indexing must deal with high variance, slow really are. During the study of the literature we made two convergence speed, weak robustness against different query observations: First, at the heart of every cracking algorithm is workloads and data distributions, and the trade-off between individual and accumulated query response time.

simple data partitioning, splitting a given key range into a certain number of partitions. Second, the main difference between individual and accumulated query response time.

In the simplest form of adaptive indexing, called database

tain number of partitions. Second, the main difference between the algorithms lies in how they distribute their indexing effort cracking or standard cracking [3], the index column is repar-titioned adaptively with respect to the incoming query pred-mostly early on, while others balance the effort as much as icates. If a range query selecting [low, high] comes in, the possible across the queries. Based on these observations, we

icates. If a range query selecting [low,high] comes in, the partition into which low falls is split into two partitions where one partitions contains all keys less than low and the other partition all keys that are greater than or equal to low. The same reorganization is repeated for the partition into which high falls. After these two steps, the range query can be answered by a simple scan of the qualifying partitions. The information which key ranges each partition holds is

E.2 Data storage representations - arrays, sorted trees.

Database index, adaptive, autonomic, query execution.

In a relational data warehouse with a hundred tables

and a thousand columns, billions of indexes are possible, in particular if partial indexes, indexes on computed columns, Permission to make digital or hard copies of all or part of this

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

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

Copyright 2010 ACM 978-1-60558-945-9/10.0003 ...\$10.00.

Keywords

1 Introduction

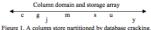
### optimized indexes

ews with their indexes are considered. ction is a central, classic, and very hard cal database design. Too few or the wrong ny queries to scan large parts of the data-dexes force high update costs. Unpredict-

ach is to tune indexes in response to the

d more often than those years ago, ex-

Database cracking [IKM 07a, KM 05] has nioneered Database cracking [IKM 07a, KM 05] has pioneered focused, incremental, automatic optimization of the representation of a data collection – the more often a key range is queried, the more its representation is optimized. This optimization occurs entirely automatically, as a side effect of queries over key ranges not yet fully optimized.



For example, after the column store illustrated in Figure 1 has been queried with range boundary values c, g, m, s, and u, all key values below c have been assigned to

### Poor Man's Sort!

Stratos Idreos Harvard University @seas.harvard.edu

the-shelf (Parallel) Mer

the-shelf (Parallel) Mergesort implemen-more expensive than a (quasi I/O bound) y three times as expensive as MonetDB's king [19]. Even though both Scanning and read and write the same amount of data, t costs. The performance difference must, putational costs: Cracking is, unlike Scan-

nted with the underlying hardware in

othesis, we make the following contributions

t an in-depth study of the contributing performance the "classic" Cracking implementation.

the findings, we develop a number of optimiza-

ased on "standard" techniques like predication, vec-on and manually implemented data parallelism using

two different parallel algorithms that exploit thread lism to make use of multiple CPU cores.

ly evaluate all developed algorithms on a number

ent systems ranging from low-end desktop machines

NU libstdc++ Version 4.8.2

ning Cracking Parallel Sorting Costs of Database Operations of Cracking is fully sorted data, its costs are Cracking is fully sorted data, its costs are so of fully sorting the data. With recent harallel) sorting algorithms [7], however, singly unattractive. To illustrate this, Figmparison of the respective operations on a 4-Core Sandy Bridge CPU.

ard Laboratories CA 94304

erbate the problem.

is to focus on enabling very fast scans, scans and columnar storage formats, an to high-bandwidth high-latency devices lisk drives and disk arrays. Low-latency uch as flash memory will likely reindex-based query processing.

ase requests and their execution plans labase requests and their execution plans, koling creation or removal of indexes on s. Such tools tend to suffer from three t, the interval between monitoring and n exceed the duration of a specific request t case there is no benefit to those tools. t is not the case, there is no index suprval, so data access during the interval ect to index creation, and eventual index an additional load that interferes with ast, but not least, traditional indexes on is equally, even if some rows are needed ver. For example, recent business trans-

treme price fluctuations are more interesting than stable prices, etc. Even where it is possible to limit an index, e.g., using a partial index or a materialized view, it is often difficult or impossible to predict the key ranges to focus on.

Column domain and storage array

20/20