INF FAKULTÄT FÜR INFORMATIK

Adaptive Indexing: between Lazy but Lightweight Cracking and Eager but Expensive Merging

Pavlo Shevchenko

Seminar on Modern Software Engineering and Database Concepts

July 13, 2017

Table of Contents

Overview of Indexing

Database Indexes:

- help to decrease the query response time;
- structure optimized for search with references to original entries;
- precalculated before the first query arrives (*offline indexing*);
- used to be stored on disk in the form of B-Trees.

Overview of Indexing

Database Indexes:

- help to decrease the query response time;
- structure optimized for search with references to original entries;
- precalculated before the first query arrives (*offline indexing*);
- used to be stored on disk in the form of B-Trees.

Issues:

- in-memory databases $\Rightarrow$ more lightweight structures;

Overview of Indexing

Database Indexes:

- help to decrease the query response time;
- structure optimized for search with references to original entries;
- precalculated before the first query arrives (*offline indexing*);
- used to be stored on disk in the form of B-Trees.

Issues:

- in-memory databases $\Rightarrow$ more lightweight structures;
- growing amounts of data $\Rightarrow$ larger indexes;

Overview of Indexing

Database Indexes:

- help to decrease the query response time;
- structure optimized for search with references to original entries;
- precalculated before the first query arrives (*offline indexing*);
- used to be stored on disk in the form of B-Trees.

Issues:

- in-memory databases $\Rightarrow$ more lightweight structures;
- growing amounts of data $\Rightarrow$ larger indexes;
- sophisticated queries $\Rightarrow$ many indexes;

Overview of Indexing

Database Indexes:

- help to decrease the query response time;
- structure optimized for search with references to original entries;
- precalculated before the first query arrives (*offline indexing*);
- used to be stored on disk in the form of B-Trees.

Issues:

- in-memory databases $\Rightarrow$ more lightweight structures;
- growing amounts of data $\Rightarrow$ larger indexes;
- sophisticated queries $\Rightarrow$ many indexes;
- unpredictable behavior of users $\Rightarrow$ hard to predicate an attribute.

Definition of Adaptive Indexing

Core Idea:

- automated maintenance and tuning of indexes by DBMS;
- continuous reorganization of the physical design during incremental *online indexing*.

Research Goal

Research Goal

Study the *evolution* of adaptive indexing:

Research Goal

Study the *evolution* of adaptive indexing:

- **Past:** Origin of adaptive indexing.

Research Goal

Study the *evolution* of adaptive indexing:

- **Past:** Origin of adaptive indexing.
- **Present:** Current approaches for adaptive indexing.

Research Goal

Study the *evolution* of adaptive indexing:

- **Past:** Origin of adaptive indexing.
- **Present:** Current approaches for adaptive indexing.
- **Future:** Unresolved questions regarding adaptive indexing.

Background. Partial Index

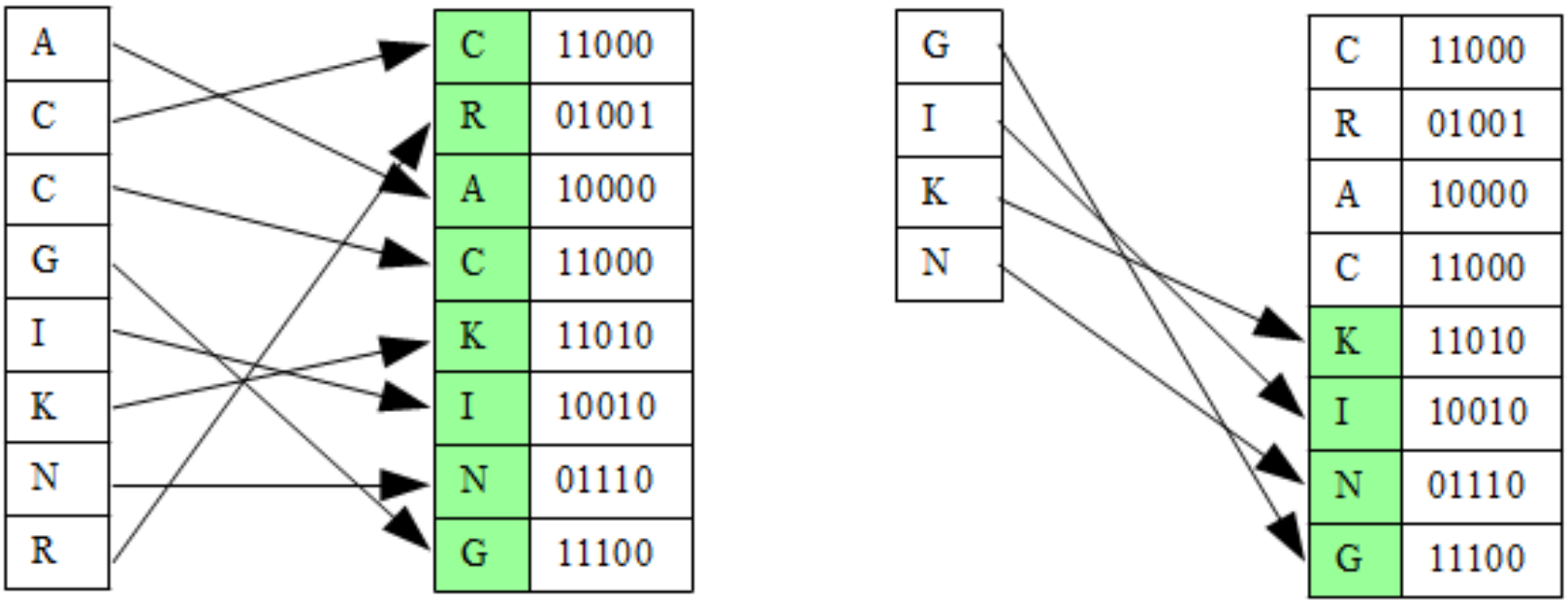

Figure: Full Index and Partial Index [1]

Background. Soft Index

In contrast to *hard indexes*, which are created and managed by DBA, *soft indexes* are created, modified and dropped by a DBMS [2].

Background. Soft Index

In contrast to *hard indexes*, which are created and managed by DBA, *soft indexes* are created, modified and dropped by a DBMS [2].

Workflow:

Background. Soft Index

In contrast to *hard indexes*, which are created and managed by DBA, *soft indexes* are created, modified and dropped by a DBMS [2].

Workflow:

1. Collect information about the current state of a system.

Background. Soft Index

In contrast to *hard indexes*, which are created and managed by DBA, *soft indexes* are created, modified and dropped by a DBMS [2].

Workflow:

1. Collect information about the current state of a system.
2. Analyze this information and choose index candidates for materialization.

Background. Soft Index

In contrast to *hard indexes*, which are created and managed by DBA, *soft indexes* are created, modified and dropped by a DBMS [2].

Workflow:

1. Collect information about the current state of a system.
2. Analyze this information and choose index candidates for materialization.
3. Create or drop some indexes based on this analysis.

Approaches for Adaptive Indexing

Goal: provide fast access to the data and the self-organized behavior of a database system.

Approaches for Adaptive Indexing

Goal: provide fast access to the data and the self-organized behavior of a database system.

1. Database Cracking [3]:
   - index maintenance is a byproduct of query processing;
   - continuous physical reorganization (*cracking* database into manageable pieces).

Approaches for Adaptive Indexing

Goal: provide fast access to the data and the self-organized behavior of a database system.

1. Database Cracking [3]:
   - index maintenance is a byproduct of query processing;
   - continuous physical reorganization (*cracking* database into manageable pieces).
2. Adaptive Merging [4]:
   - adaptive nature of cracking;
   - better query response time and quicker adaption.

Approaches for Adaptive Indexing

Goal: provide fast access to the data and the self-organized behavior of a database system.

1. Database Cracking [3]:
   - index maintenance is a byproduct of query processing;
   - continuous physical reorganization (*cracking* database into manageable pieces).
2. Adaptive Merging [4]:
   - adaptive nature of cracking;
   - better query response time and quicker adaption.
3. Hybrid Approaches [5]:
   - combine strength of *database cracking* and *adaptive merging*.

Database Cracking

Main Idea: incrementally perform quicksort on a copy of a column using crack-in-three or crack-in-two.

Database Cracking

Main Idea: incrementally perform quicksort on a copy of a column using crack-in-three or crack-in-two.

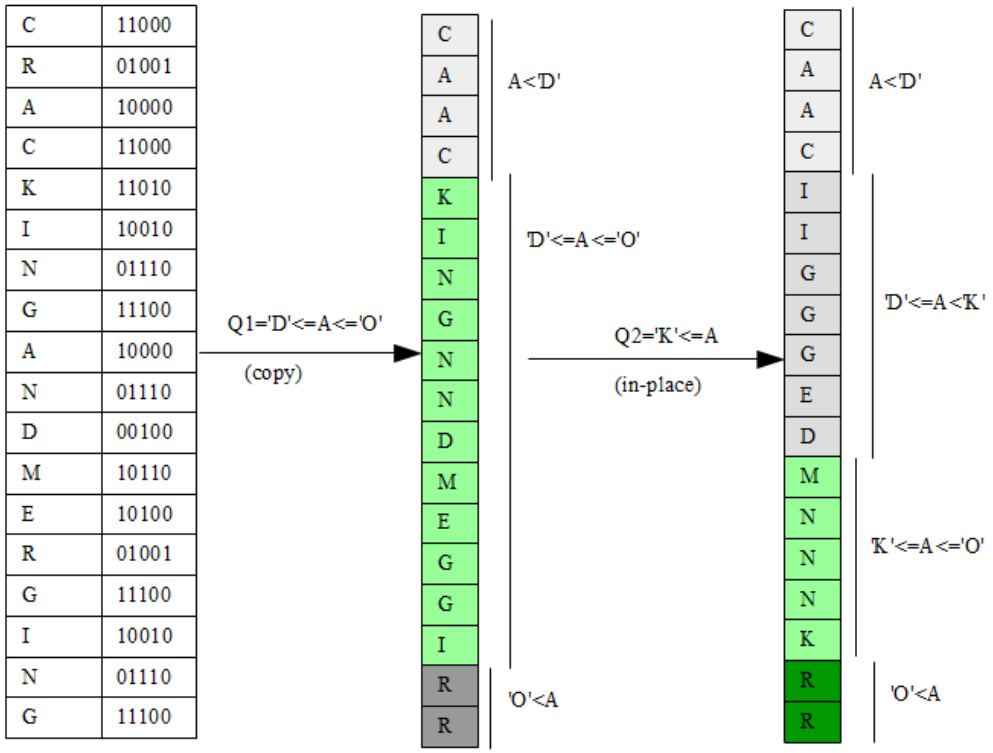

Figure: Cracking a column.

Adaptive Merging

Main Idea: incrementally merge sort the relevant key ranges till a single sorted partition is created.

Adaptive Merging

Main Idea: incrementally merge sort the relevant key ranges till a single sorted partition is created.

C	11000
R	01001
A	10000
C	11000
K	11010
I	10010
N	01110
G	11100
A	10000
N	01110
D	00100
M	10110
E	10100
R	01001
G	11100
I	10010
N	01110
G	11100

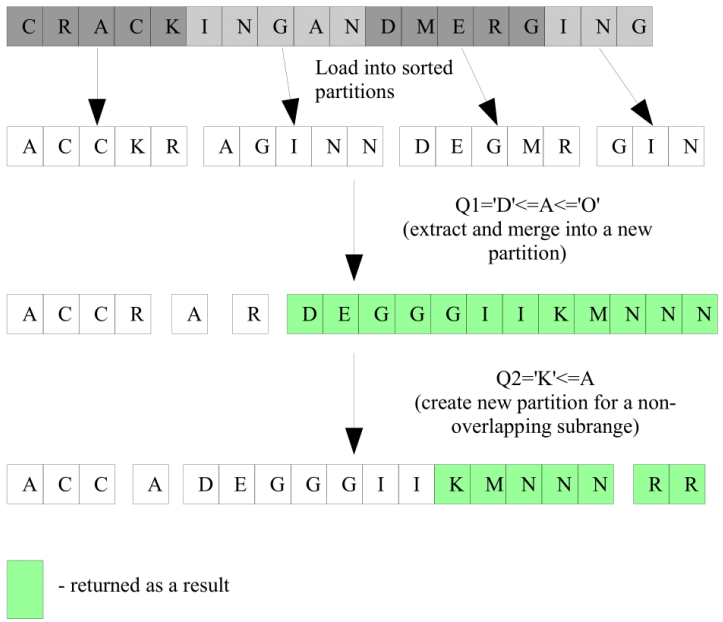

Figure: Example for adaptive merging.

Database Cracking vs. Adaptive Merging

Initial Run
Response Time
Convergence

Database Cracking vs. Adaptive Merging

	Database Cracking
Initial Run	slightly slower than scan
Response Time	faster than scan after 1^{st} query
Convergence	after 1k queries, 40% slower than full index

Database Cracking vs. Adaptive Merging [1]

	Database Cracking	Adaptive Merging
Initial Run	slightly slower than scan	approx. 5 times slower than scan
Response Time	faster than scan after 1^{st} query	first queries are slower than scan
Convergence	after 1k queries, 40% slower than full index	10M records $\Rightarrow$ approx. 40 queries

[1] experimental data taken from [5]

Hybrid Approaches

Strategy

Hybrid Approaches

Strategy

1. Find the general structure:
   - load data into partitions;
   - extract relevant data from each partition (*crack*);
   - load data into final partitions.

Hybrid Approaches

Strategy

1. Find the general structure:
   - load data into partitions;
   - extract relevant data from each partition (*crack*);
   - load data into final partitions.
2. How to make components better?
   - Crack;
   - Sort;
   - Radix.

Hybrid Approaches

Strategy

1. Find the general structure:
   - load data into partitions;
   - extract relevant data from each partition (*crack*);
   - load data into final partitions.
2. How to make components better?
   - Crack;
   - Sort;
   - Radix.
3. Combine the best components
   - Sort-*;
   - Crack-*;
   - Radix-*.

Crack-Sort Algorithm

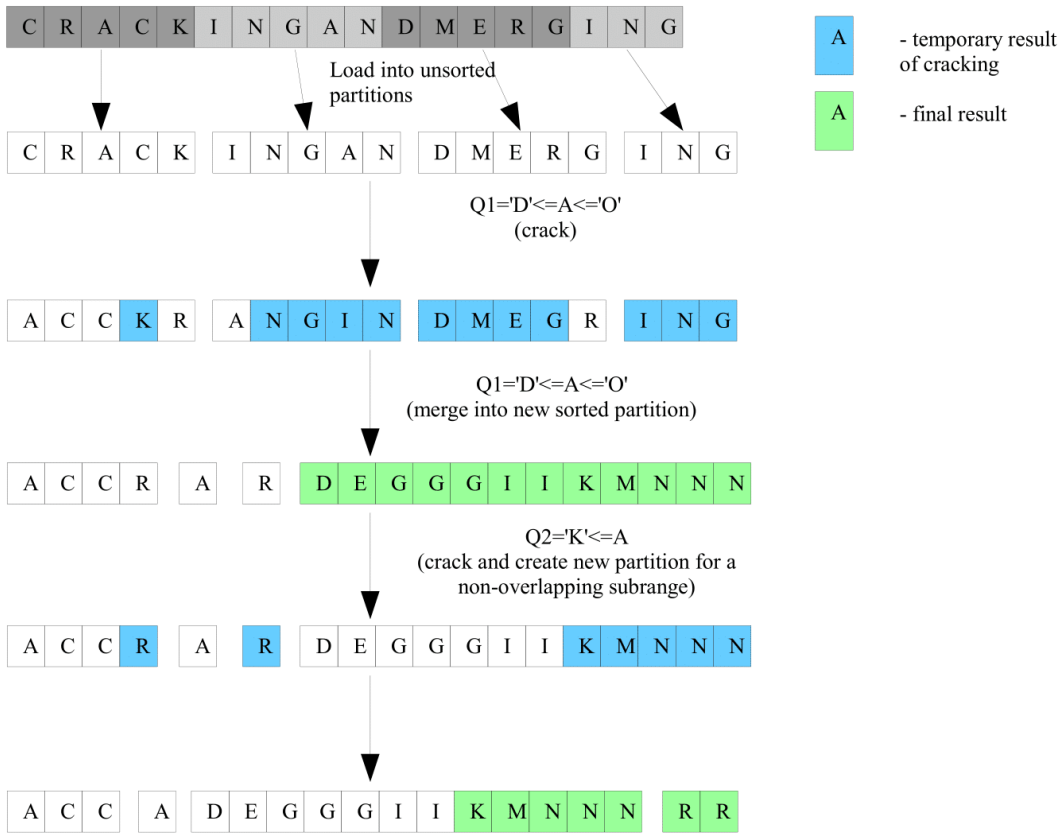

Figure: Hybrid Crack-Sort Algorithm.

Radix-Radix Algorithm

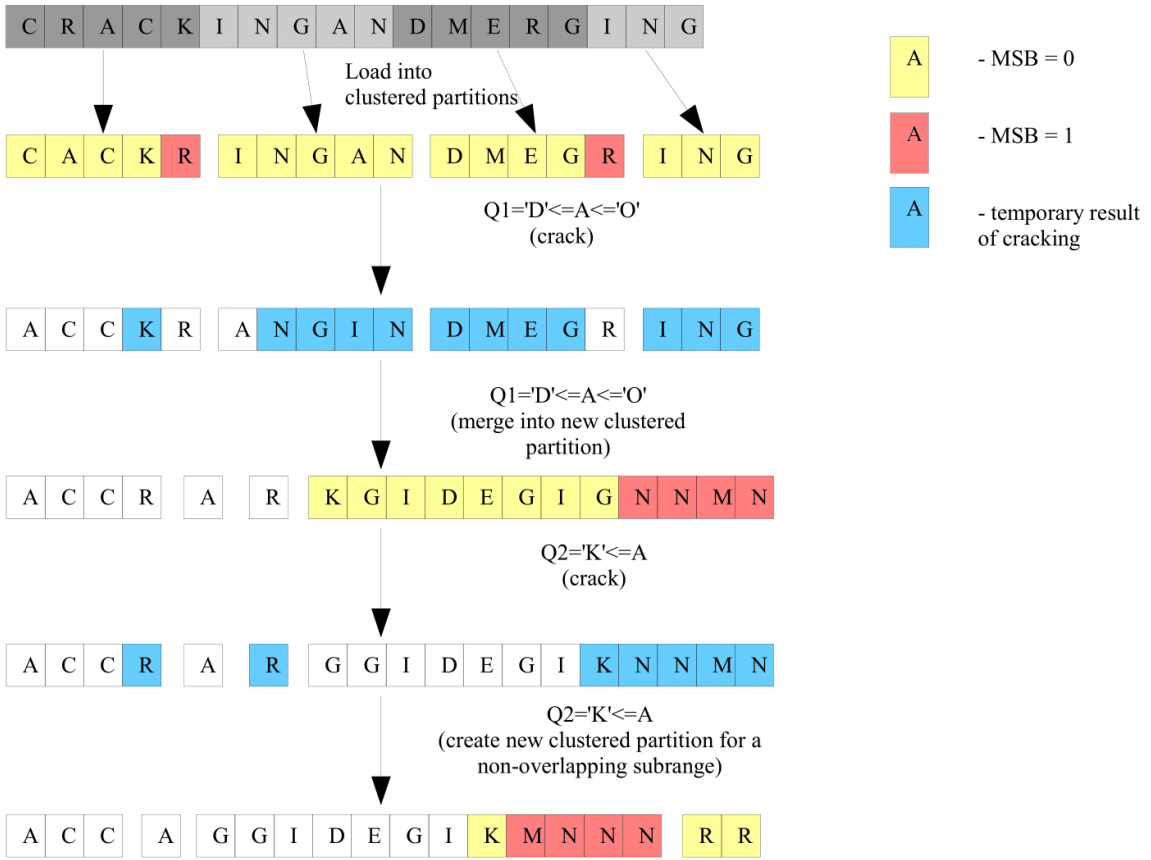

Figure: Hybrid Radix-Radix Algorithm

Lessons Learned

- Adaptive Indexing Is a Promising Concept;
- Database Cracking and Adaptive Merging Have Similar Nature;
- Crack-Sort and Crack-Radix Are Promising Alternatives;
- Hybrid Algorithms Can Be Further Combined.

Future Work

- Further Research on Adaptive Merging and Hybrid Approaches Needed;
- New Strategies for Hybrid Algorithms;
- Influence of Underlying Data Structure;
- Integration of Adaptive Indexing into Other Models;
- Extensions to the Methods.

Thank you for your attention!

Questions? Recommendations? Remarks?

Literature

1. Seshadri, Praveen, and Arun Swami. Generalized partial indexes. Data Engineering, 1995. Proceedings of the Eleventh International Conference on. IEEE, 1995.
2. Lühring, Martin, et al. Autonomous management of soft indexes. Data Engineering
3. Idreos, Stratos, Martin L. Kersten, and Stefan Manegold. Database Cracking. CIDR. Vol. 7. 2007.
4. Graefe, Goetz, and Harumi Kuno. Self-selecting, self-tuning, incrementally optimized indexes. Proceedings of the 13th International Conference on Extending Database Technology. ACM, 2010.
5. Idreos, Stratos, et al. Merging whats cracked, cracking whats merged: adaptive indexing in main-memory column-stores. Proceedings of the VLDB Endowment 4.9, 2011.