ADAPTIVITY IN DATABASE KERNELS

Adaptive Indexing: Self tuning access methods

Javam Machado   Elvis Teixeira   Paulo Amora

25 de agosto de 2017

Universidade Federal do Ceará - UFC

Universidade Federal do Ceará

lsbd

SUMARY

- RECAP
  - New Problems
  - Adaptive vs Offline
- Database Cracking
  - Adaptive index for column stores
- Adaptive merging
  - Adaptive index for tuple based storage
- Concurrency

RECAP

FAST DATA, URGENT NEED FOR INSIGHTS

Data must be processed at least as quickly as it is produced!

Data layout must be flexible and specialized to the workload.

Tuning must be autonomous.

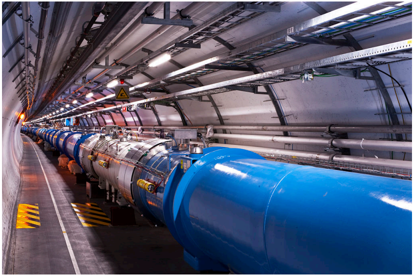


Figure: The Large Hadron Collider

In many modern applications e. g. **big data exploration**, the query pattern is unknown until it is actually processed

Data is produced continuously, there is no time to fully optimize physical layout (offline tuning)

Fast and large data analysis strategies:

- Scalable
- Distributed
- Comodity hardware
- Map Reduce

Figure: Apache Hadoop

Heterogeneous

Social

Autonomous

Figure: SETI @ Home

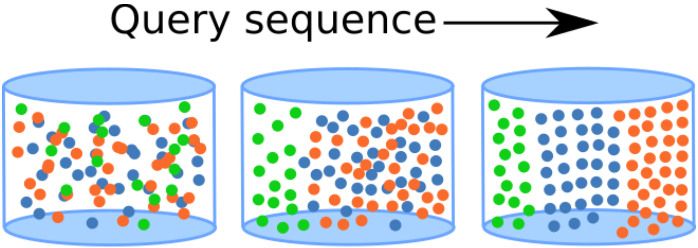
What about DBMS?

Offline indexes

- Require a decision on what to index
- One step operation (CREATE INDEX, DROP INDEX)
- Changes in workload demand rebuild

Adaptive indexes

- Index selection is made on first query
- Physical design is tuned by incremental actions
- Changes occur in response to current query
- Changes in workload are naturally handled


Query sequence

DATABASE CRACKING

Developed for column stores (MonetDB)

Partitions an attribute at each query

In memory column copy and supporting AVL tree

Zero initialization

select ... where A >= 6;

2
0
1
3
4
9
6
8
7
5

```
algorithm CrackInTwo(Low, High, Med)
    x1 := point at position Low
    x2 := point at position High
    while position(x1) < position(x2) do
        if value(x1) < Med then
            x1 := point at next position
        else
            while value(x2) >= Med and
            position(x2) > position(x1) do
                x2 := point at previous position
            end while
            Exchange(x1, x2)
            x1 := point at next position
            x2 := point at previous position
        end if
    end while
```

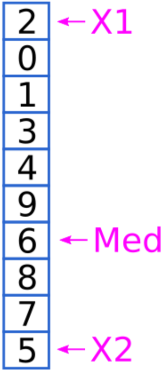
Idreos et. al. 2007 - Database Craking

```
algorithm CrackInTwo(Low, High, Med)
    x1 := point at position Low
    x2 := point at position High
    while position(x1) < position(x2) do
        if value(x1) < Med then
            x1 := point at next position
        else
            while value(x2) >= Med and
            position(x2) > position(x1) do
                x2 := point at previous position
            end while
            Exchange(x1, x2)
            x1 := point at next position
            x2 := point at previous position
        end if
    end while
```

Idreos et. al. 2007 - Database Craking

CRACKING COLUMN

select ... where A >= 6;

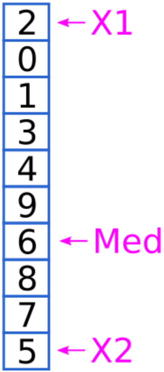


```
algorithm CrackInTwo(Low, High, Med)
    x1 := point at position Low
    x2 := point at position High
    while position(x1) < position(x2) do
        if value(x1) < Med then
            x1 := point at next position
        else
            while value(x2) >= Med and
            position(x2) > position(x1) do
                x2 := point at previous position
            end while
            Exchange(x1, x2)
            x1 := point at next position
            x2 := point at previous position
        end if
    end while
```

Idreos et. al. 2007 - Database Craking

CRACKING COLUMN

select ... where A >= 6;



select ... where A >= 6;

2	
0	
1	
3	
4	
9	← X1
6	← Med
8	
7	
5	← X2

```
algorithm CrackInTwo(Low, High, Med)
    x1 := point at position Low
    x2 := point at position High
    while position(x1) < position(x2) do
        if value(x1) < Med then
            x1 := point at next position
        else
            while value(x2) >= Med and
            position(x2) > position(x1) do
                x2 := point at previous position
            end while
            Exchange(x1, x2)
            x1 := point at next position
            x2 := point at previous position
        end if
    end while
```

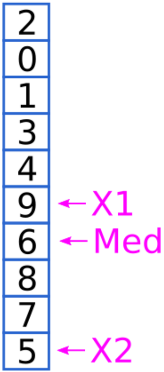
Idreos et. al. 2007 - Database Craking

select ... where A >= 6;

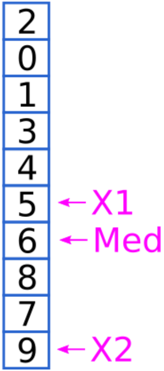
```
algorithm CrackInTwo(Low, High, Med)
    x1 := point at position Low
    x2 := point at position High
    while position(x1) < position(x2) do
        if value(x1) < Med then
            x1 := point at next position
        else
            while value(x2) >= Med and
            position(x2) > position(x1) do
                x2 := point at previous position
            end while
            Exchange(x1, x2)
            x1 := point at next position
            x2 := point at previous position
        end if
    end while
```

Idreos et. al. 2007 - Database Craking

select ... where A >= 6;

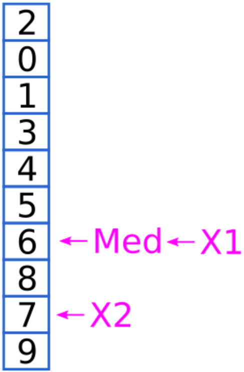


select ... where A >= 6;



CRACKING COLUMN

select ... where A >= 6;

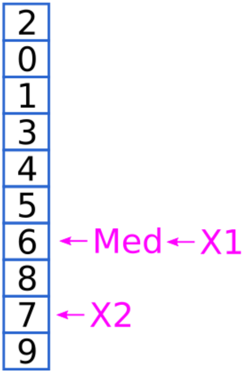


UNIVERSIDADE FEDERAL DO CEARÁ

```
algorithm CrackInTwo(Low, High, Med)
    x1 := point at position Low
    x2 := point at position High
    while position(x1) < position(x2) do
        if value(x1) < Med then
            x1 := point at next position
        else
            while value(x2) >= Med and
            position(x2) > position(x1) do
                x2 := point at previous position
            end while
            Exchange(x1, x2)
            x1 := point at next position
            x2 := point at previous position
        end if
    end while
```

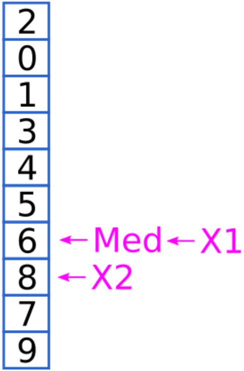
Idreos et. al. 2007 - Database Craking

select ... where A >= 6;

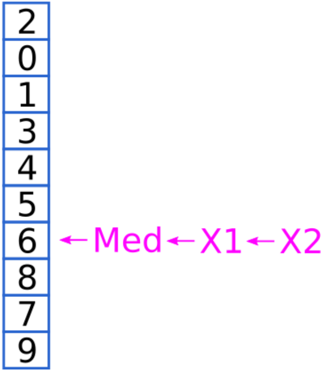


UNIVERSIDADE FEDERAL DO CEARÁ

select ... where A >= 6;

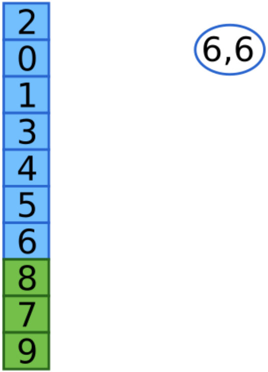


select ... where A >= 6;

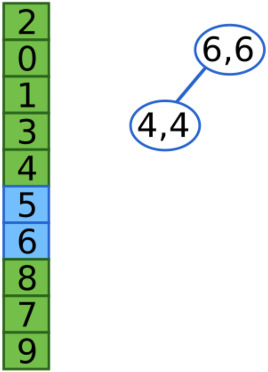


Partitions are stored in a tree structure (cracker index)

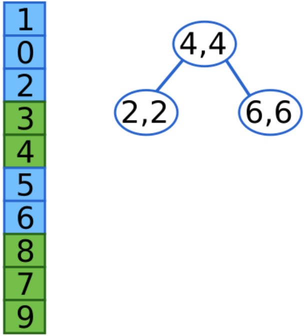
Partitions are stored in a tree structure (cracker index)



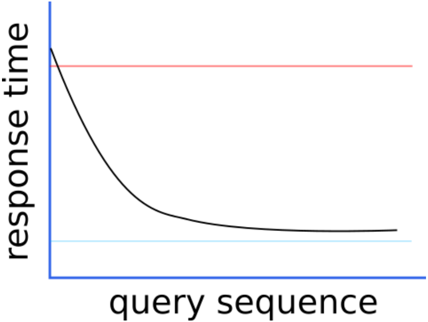
More queries - more partitions - smaller pieces scanned



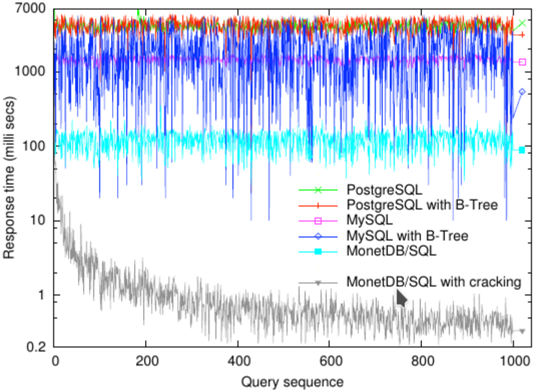
More queries - more partitions - smaller pieces scanned



Response times are expected to decrease from the level of full scans ($O(N)$) to near the level of a binary search ($O(log(n))$)



Database cracking - response times



Idreos et. al. 2007 - Database Cracking

A histogram for free [1]

Column partitions contain information on the distribution of the data attribute. i. e. they tell how many records lie in the given range.

[1] Idreos et. al. 2007 - Database Craking

Cracking aided joins [2]

The same histogram-like information can be used to exclude partitions to consider while executing joins.

[2] Idreos et. al. 2007 - Database Craking

Stochastic cracking

- Partition ranges are not equal to query ranges
- Adds a random component to cracking
- Eventually cracks big partitions

Holistic indexing

- Idle CPU cores are used to perform cracks
- Select operators still perform cracks
- Holistic cracks are performed on the biggest partitions

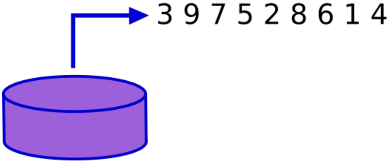
ADAPTIVE MERGING

Relational systems are typically stored in disk

B-tree based structures are suitable for block storage

Full sorting may be prohibitive (time)

And demands prior index selection (workload knowledge)


3 9 7 5 2 8 6 1 4

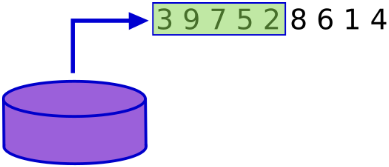


Figure: Collect run

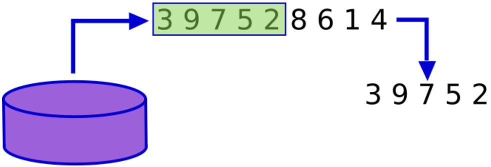


Figure: Collect run

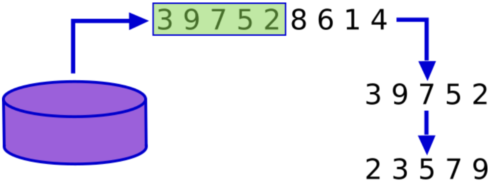


Figure: Sort run

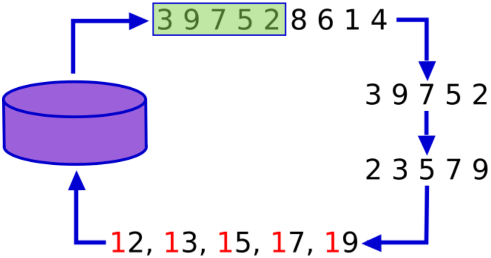


Figure: Add partition key

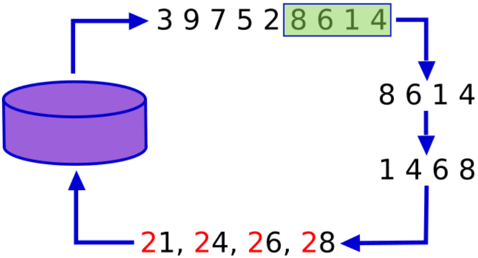


Figure: Repeat for other partitions

12, 13, 15, 17, 19, 21, 24, 26, 28

Figure: Final sorted data

Structure creation

- Runs become the data in the leaf level of a B+ tree
- A bulk load procedure is used to build the tree

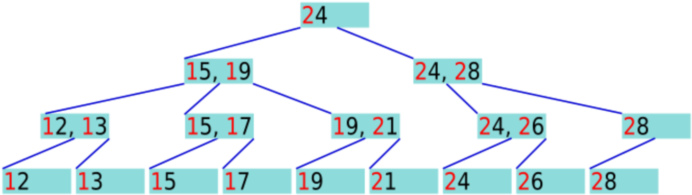


Figure: Complete tree

SELECT * FROM t WHERE t.A > 5 AND t.A <= 7;

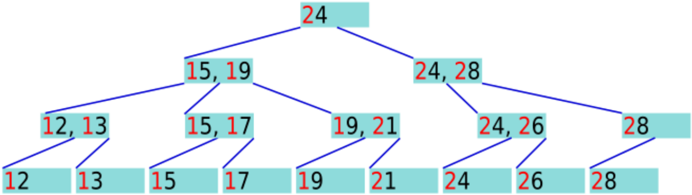


Figure: Answering a query

SELECT * FROM t WHERE t.A > 5 AND t.A <= 7;

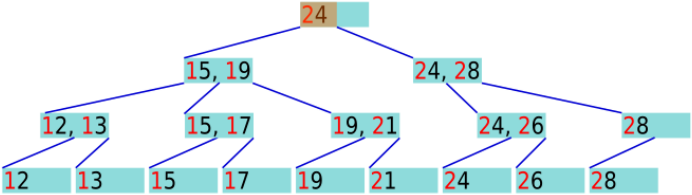


Figure: Answering a query

SELECT * FROM t WHERE t.A > 5 AND t.A <= 7;

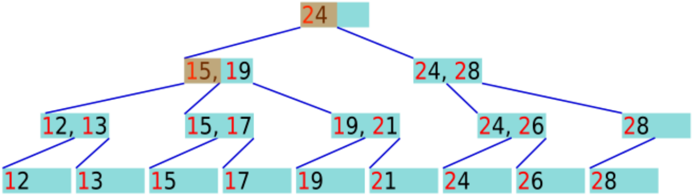


Figure: Answering a query

SELECT * FROM t WHERE t.A > 5 AND t.A <= 7;

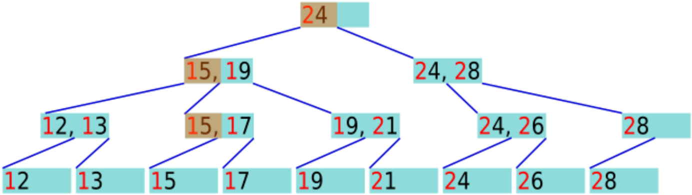


Figure: Answering a query

SELECT * FROM t WHERE t.A > 5 AND t.A <= 7;

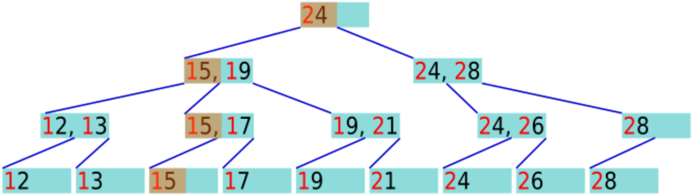


Figure: Answering a query

SELECT * FROM t WHERE t.A > 5 AND t.A <= 7;

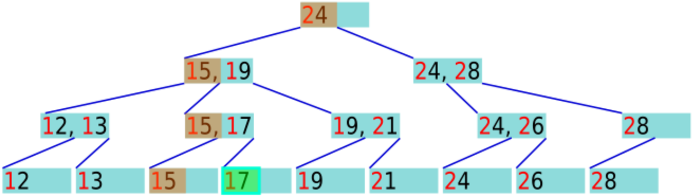


Figure: Answering a query

SELECT * FROM t WHERE t.A > 5 AND t.A <= 7;

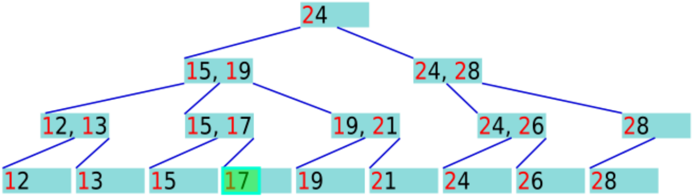


Figure: Answering a query

SELECT * FROM t WHERE t.A > 5 AND t.A <= 7;

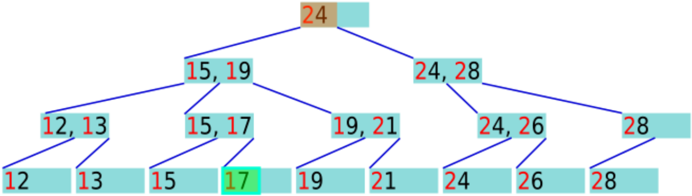


Figure: Answering a query

SELECT * FROM t WHERE t.A > 5 AND t.A <= 7;

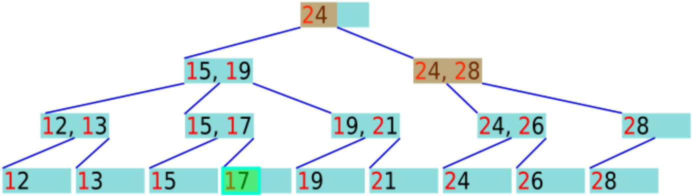


Figure: Answering a query

SELECT * FROM t WHERE t.A > 5 AND t.A <= 7;

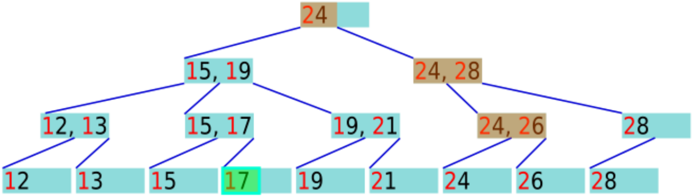


Figure: Answering a query

SELECT * FROM t WHERE t.A > 5 AND t.A <= 7;

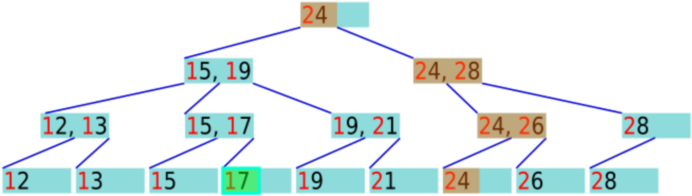


Figure: Answering a query

SELECT * FROM t WHERE t.A > 5 AND t.A <= 7;

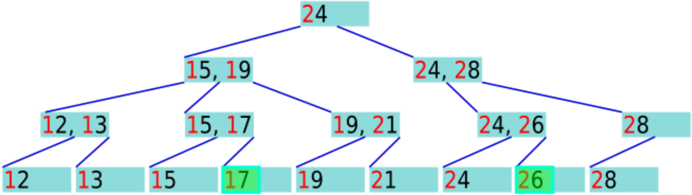


Figure: Answering a query

Each query walks the tree and move the qualifying tuples to the final partition

SELECT * FROM t WHERE t.A > 5 AND t.A <= 7;

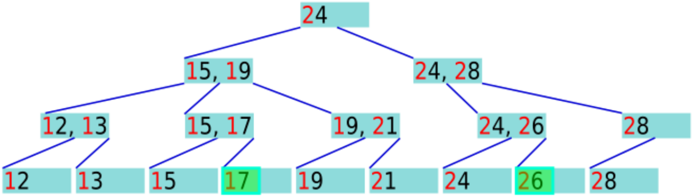


Figure: Adaptive Merging

SELECT * FROM t WHERE t.A > 5 AND t.A <= 7;

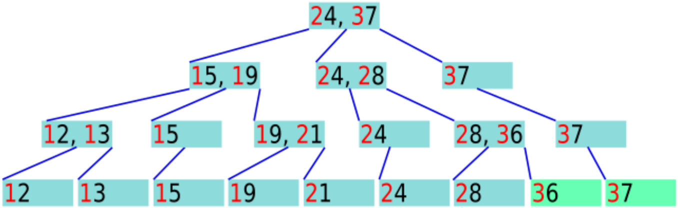


Figure: Short Query Ranges

Adaptive Merging - overhead per query

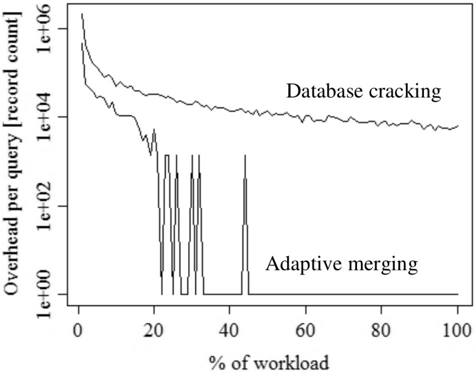


Figure: Short Query Ranges

Grafe et. al. 2010 - Self-selecting, self-tuning incrementally optimized indexes

Adaptive Merging - overhead per query

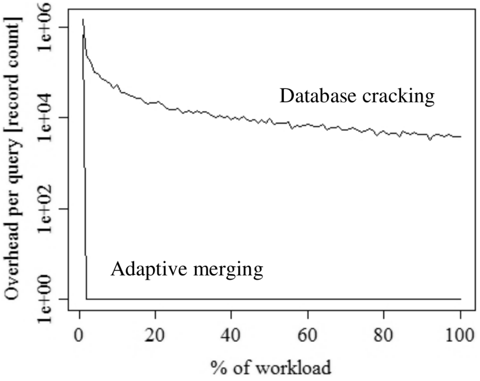


Figure: Long Query Ranges

Grafe et. al. 2010 - Self-selecting, self-tuning incrementally optimized indexes

CONCURRENCY

The problem

Updating index structures while processing queries requires concurrency control and the system may incur additional lock contention

Index structure VS index contents [3]

- Index logical contents do not change
- Index refinement is not transactional
- Lightweight latches instead of locks

[3] Graefe et. al. 2012 - Concurrency Control for Adaptive Indexing

Locks VS Latches

	Locks	Latches
Separate	Transactions	Threads
Protect	DB Content	In-memory data
During	Entire Transactions	Critical sections

Incremental granularity of locking [4]

- Increasingly smaller key ranges affected
- Conflicts can be avoided

[4] Graefe et. al. 2012 - Concurrency Control for Adaptive Indexing

AI/ML guided layout optimization

- Incremental physical layout tuning enables learning
- Current request X Workload pattern
- Workload forecasting (tune in anticipation)

Flexible physical design tuning

Autonomous

Enable the use of workload pattern recognition

Fits modern query processing

Questions?