ADAPTATIVE INDEXES

Incrementally built self tuning access methods

Javam Machado Elvis Teixeira Paulo Amora

25 de agosto de 2017

Universidade Federal do Ceará - UFC

Universidade Federal do Ceará

ADAPTIVE INDEXING

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

Ex: LHC produced 25PB a year by 2012, needs the largest computing grid today (2017), Worldwide LHC Computing Grid with more than 170 computing centers worldwide.

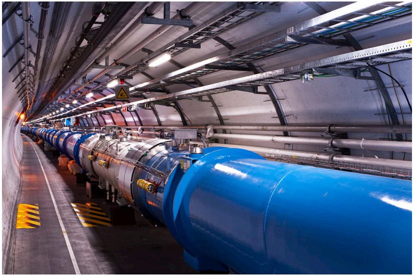


Figure: The large hadron collider

Fast large data analysis strategies

Figure: Apache Hadoop

VOLUNTEER COMPUTING

Figure: SETI @ Home

In many modern applications e. g. **big data exploration**, the query pattern is not known before it is actually processed.

Indexes and physical design in general take time to be properly tuned. No query can be answered during tuning times.

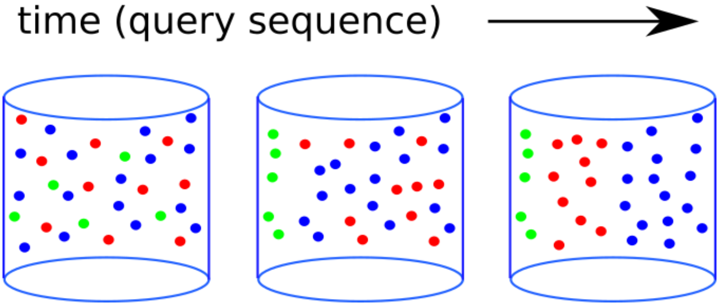
Offline indexes

- 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

ADAPTIVE INDEXING CHANGES PHYSICAL LAYOUT



DATABASE CRACKING

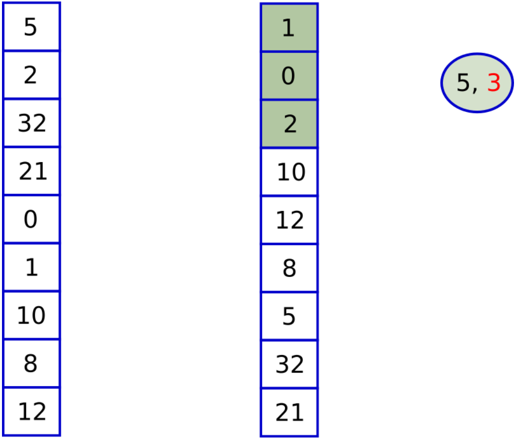
Developed for column stores

Partitions an attribute at each query

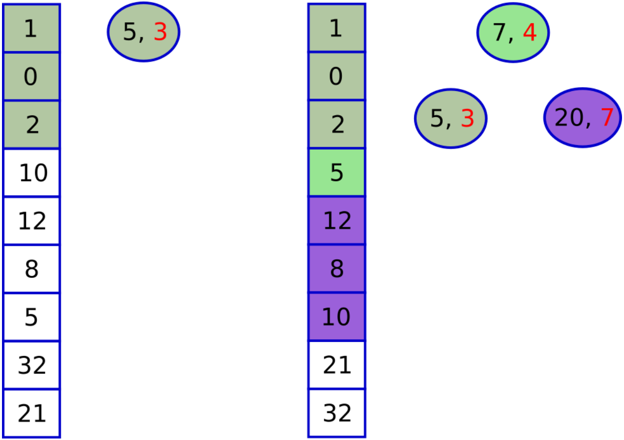
In memory column copy and supporting AVL tree

Zero initialization

SELECT * FROM t WHERE t.A < 5;



SELECT * FROM t WHERE t.A > 7 AND t.A <= 20;

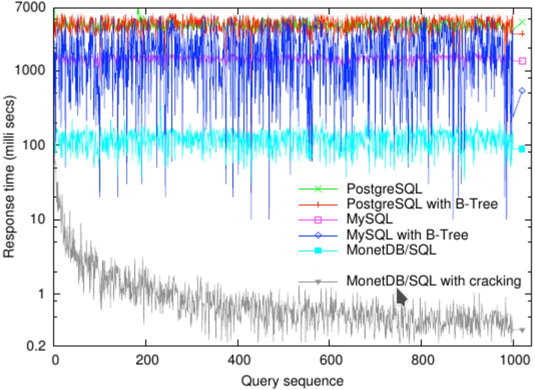


```
algorithm CrackInTwo(Low, High, Med)
    x1 := point at position Low
    x2 := point at position High
    while position(x1) < position(x2) do
        if value(x1) < Med then
            x1 := point at next position
        else
            while value(x2) >= Med and
            position(x2) > position(x1)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

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

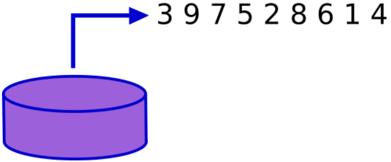
BLOCK STORAGE

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

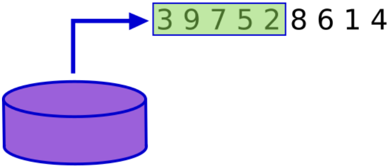


Figure: Collect run

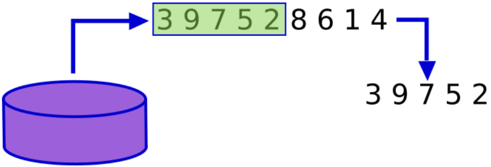


Figure: Collect run

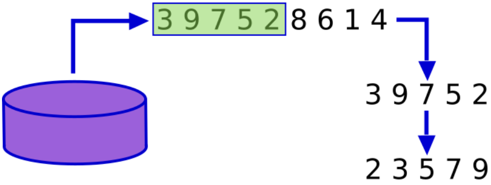


Figure: Sort run

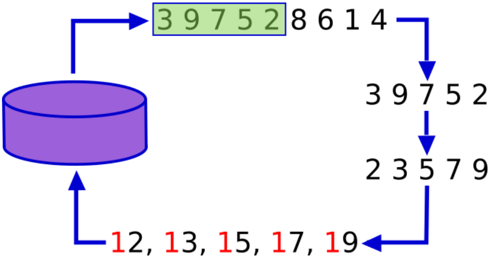


Figure: Add partition key

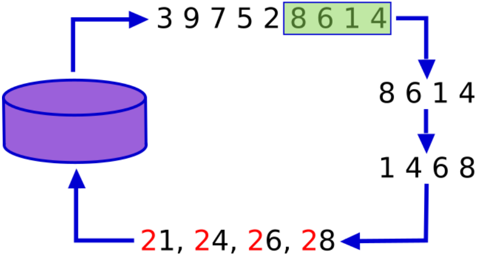


Figure: Repeat for other partitions

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

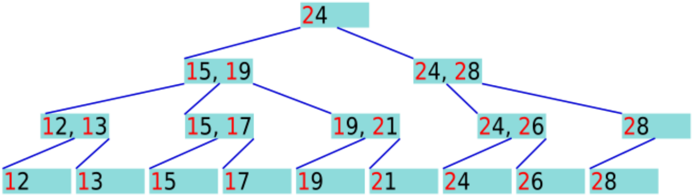


Figure: Complete tree

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

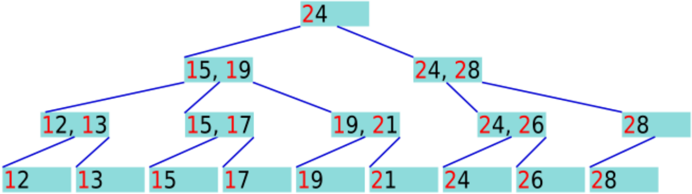


Figure: Answering a query

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

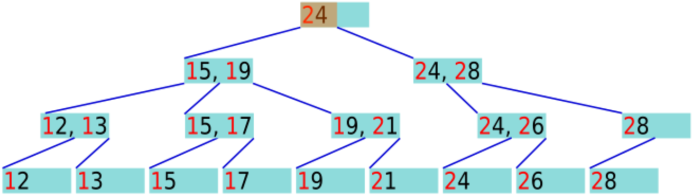


Figure: Answering a query

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

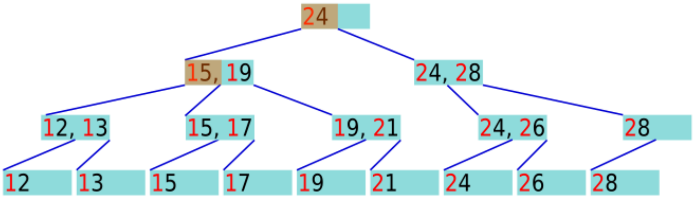


Figure: Answering a query

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

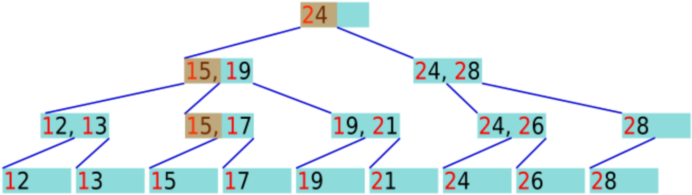


Figure: Answering a query

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

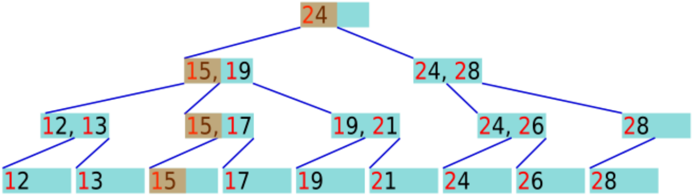


Figure: Answering a query

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

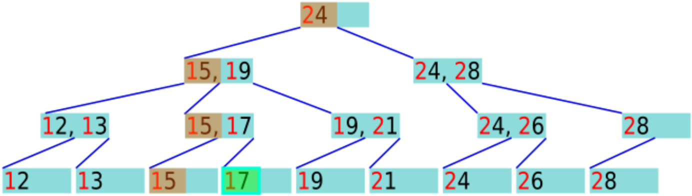


Figure: Answering a query

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

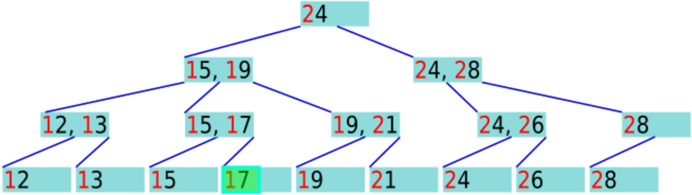


Figure: Answering a query

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

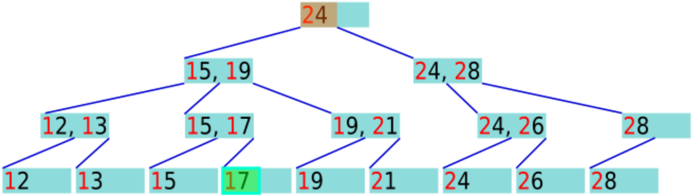


Figure: Answering a query

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

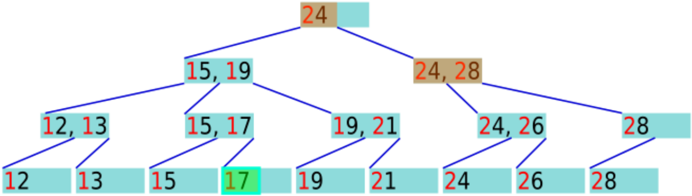


Figure: Answering a query

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

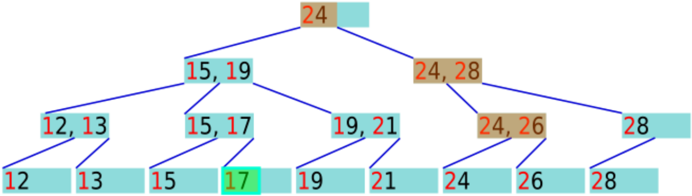


Figure: Answering a query

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

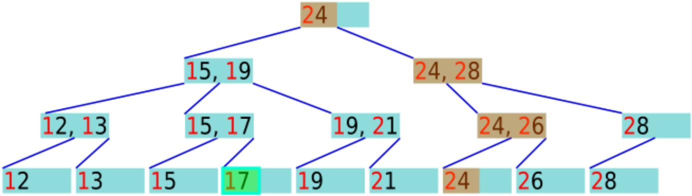


Figure: Answering a query

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

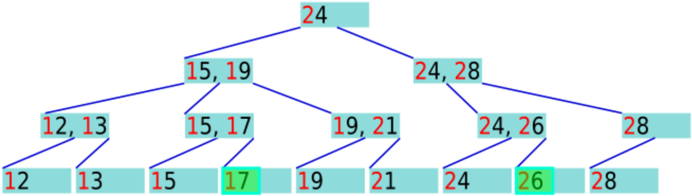


Figure: Answering a query

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;

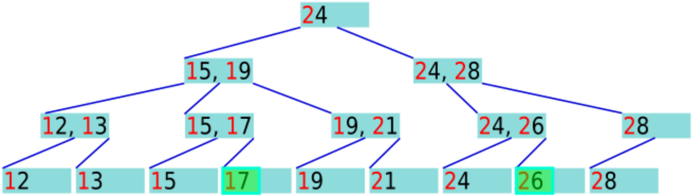


Figure: Adaptive Merging

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

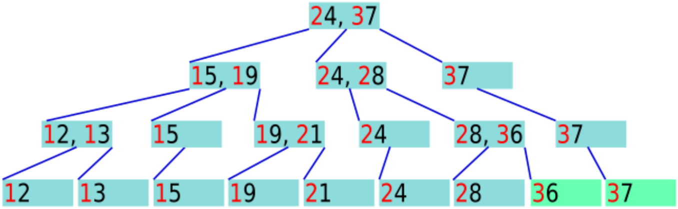


Figure: Short Query Ranges

Adaptive Merging - overhead per query

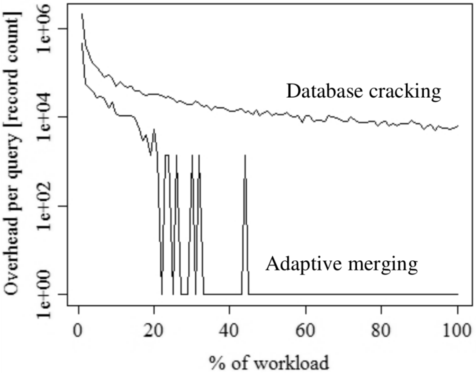


Figure: Short Query Ranges

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

Adaptive Merging - overhead per query

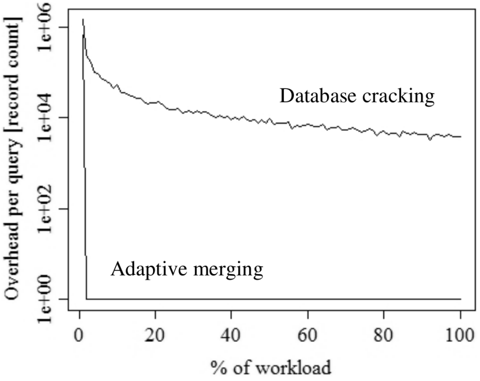


Figure: Long Query Ranges

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

Adaptive Indexing

- Incremental changes in physical layout
- Structure changes based on the current workload
- Concurrency issues also improves adaptively

Questions?