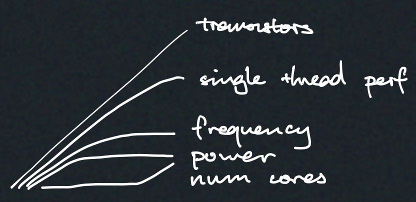
Lec 13 Query Execution I

Hardware trends



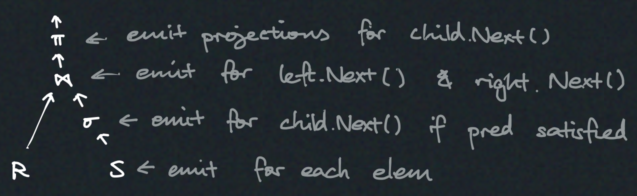
Processing Models

Defines how the system executes query plan

▹ Iterator Model aka Volcano, Pipeline

Old model, didn't think about L1, L2, etc.

Next(), Open(), Close() for each operator
- └ end
- └ start
- └ iterator for getting output


π ← emit projections for child.Next()
⋈ ← emit for left.Next() & right.Next()
σ ← emit for child.Next() if pred satisfied
R
S ← emit for each elem

- Lots of function calls
- Access not batched
+ Easy to implement

▹ Materialisation Model

Instead of child.Next() iterator, do child.Output() to get entire child result.

+ Fewer function calls
+ Lower overhead
+ Good for OLTP
- Bad for OLAP due to large intermediate result

▹ Vectorisation Model — batches of tuples

Next() returns a batch, some 2K of tuples

+ Batches more likely to stay in processor cache
+ Good for OLAP

* Flow direction
  - Bottom up
  - Top down

Access Methods

How to access data stored in table

▹ Sequential scan

DBMS maintains cursor

Usually the worse, but often the only way

▹ Data skipping
- → Lossy, approximate result
- → Zone map — pre-computed aggregation for each page
  - └ eg. skip page if out of min/max of page
  - └ storage
    - → on the page
      - still need to access the page
    - → elsewhere
      - have to maintain it

▹ Index scan
- └ Pick an index (ideally the most selective index) to use to find records
  - → Use summary histogram to estimate tuple count given predicate

- → Multi-index scan
  - └ filter on multiple indices, then set operations
  - may be bad if indices correlated

Modification Queries

insert, delete, update

Problem: these can mess up the index being scanned
└ The Halloween problem (an actual IBM bug)

Solutions
- → Materialisation
- → Track modified records per query

* Expression evaluation

- → Just evaluate the tree
- → JIT compilation (ask LLVM to generate assembly)
  - → PREPARE to do JIT for template to asm with param
    eg. PREPARE xxx AS select * from S where S.val = $1 +9