more about column-store plans and compression

prof. Stratos Idreos

HTTP://DASLAB.SEAS.HARVARD.EDU/CLASSES/CS165/
query compilation

an “ancient” yet new topic/research challenge

query->SQL->interpret to a generic query plan->c functions

query->SQL->generate and link tailored c code
query compilation

an “ancient” yet new topic/research challenge

query->SQL->interpret to a generic query plan->c functions

query->SQL->generate and link tailored c code

why this can be better/faster
Generating code for holistic query evaluation
Konstantinos Krikellias, Stratis Viglas, Marcelo Cintra:
In Proc. of the Inter. Conference on Data Engineering (ICDE), 2010

Efficiently Compiling Efficient Query Plans for Modern Hardware
Thomas Neumann
In Proc. of the Very Large Databases Conference (VLDB), 2011

H2O: A Hands-free Adaptive Store
Ioannis Alagiannis, Stratos Idreos, and Anastassia Ailamaki

Data Blocks: Hybrid OLTP and OLAP on Compressed Storage using both Vectorization and Compilation
Harald Lang, Tobias Mühlbauer, Florian Funke, Peter A. Boncz, Thomas Neumann, Alfons Kemper
research night coming up
during Thursday labs next week

can we have bitmap indexing that works great for both reads and writes?

UpBit: Scalable In-Memory Updatable Bitmap Indexing
M. Athanassoulis, Z. Yan, and S. Idreos
working over fixed width & dense columns

**select**

```c
for (i=0; i<size; i++)
    if column[i] > v
        res[j++] = i
```

no function calls, no indirections, no auxiliary data, min ifs
easy to prefetch next data values

**fetch**

```c
for (i=0; i<size; i++)
    inter2[j++] = column[inter1[i]]
```
select min(C) from R where A<10 & B<20
The figure illustrates a query plan with the following components:

- **CPU**
- **Registers**
- **On-chip cache**
- **On-board cache**
- **Memory**
- **Disk**

The query plan is depicted with operations labeled as `op1`, `op2`, and `op3`, with nodes A and B connected in a sequential flow from A to B to A.

- **Faster** and **Cheaper** labels indicate performance and cost attributes for each level in the hierarchy.

The diagram suggests a trade-off between speed (faster) and cost (cheaper) across different storage and processing layers.
tuple at a time - good for minimizing memory footprint
bulk processing - good minimizing functional overhead
vectorized processing - somewhere in between
~1960s

tuple at a time

1980s: ideas about block processing

tuple at a time

2005: vectorwise

2010: industry adoption

history/timeline
project: column-at-a-time

bonus: vectorized processing
update row7=\{(A=a, B=b, C=c, D=d)\}

cost: 1 page

cost: \(N\) pages, \(N=\#\) of columns
A

B

C

D

base data

query

periodically

A

B

C

D

pending updates

update

A

B

C

D
pending inserts  pending deletes

update = delete followed by insert

what information do we need to remember
Compression = data & computation

- CPU
- Registers
- On-chip cache
- On-board cache
- Memory
- Disk

Compute

Data

Speed

Time

But

CPU

Mem
8 bytes width

value1
value2
value3
value1
value1
value4
value2
value3
value5
...

3 bits width

001
010
011
001
001
100
010
011
101
...

8 bytes width

value1
value2
value3
value4
value5
...

how many bits do we need for the codes (width)
which one gives better compression
and how do we process data
e.g., select A<10
3 bits width
001
010
011
001
001
100
010
011
101
...

8 bytes width
value1
value2
value3
value4
value5

think about billions of entries…

can we do any better
can we do something like huffman coding any side-effects

(check: Business Analytics in (a) Blink from readings)
fixed-width is key…
ok and how do we store variable length data
fixed-width is key…
ok and how do we store variable length data

dictionary of strings

fixed width codes that point to dictionary entries
compression is a bonus task for the project

Extra: Joins on Encoded and Partitioned Data
Jae-Gil Lee, Gopi K. Attaluri, Ronald Barber, Naresh Chainani, Oliver Draese, Frederick Ho, Stratos Idreos, Min-Soo Kim, Sam Lightstone, Guy M. Lohman, Konstantinos Morfonios, Keshava Murthy, Ippokratis Pandis, Lin Qiao, Vijayshankar Raman, Vincent Kulanndai Samy, Richard Sidle, Knut Stolze, Liping Zhang
In Proc. of the Very Large Databases Conference (VLDB), 2014

IBM Blue
late tuple reconstruction/materialization
only reconstruct to present results

no need to assemble tuples
minimize memory footprint
minimize data we are moving up the memory hierarchy
but requires new processing engine
Assume a column-store database with a table R(A,B,C,D,E). All attributes are integers. Our workload has two classes of queries:

1) select max(B),max(C),max(D),max(E) from R where A>v1
2) select B+C+D+E from R where A>v1

Should we use late or early tuple reconstruction plans? For each query, draw the 2 possible plans and the respective operators, explain which one is best and give the total cost.
### Late TR

#### Hybrid

**sel A**

**IDs**

**B**

**max**

**IDs**

**C**

**max**

**IDs**

**D**

**max**

**result**

**late TR**

**max(B), max(C)**

**max(D), max(E)**

**result**

**hybrid**
sel A → IDs B → max → IDs C → max → IDs D → max

result

late TR

sel A → IDs B → C → D → E → max(B), max(C) max(D), max(E)

result

hybrid
sel A IDs B max sel A IDs C max sel A IDs D max
result 0000

late TR

max(B), max(C) max(D), max(E)
result

hybrid
sel A → IDs B → max C → IDs D → max E

result 0000

late TR

sel A → IDs B → C → D → E

max(B), max(C), max(D), max(E)

result

hybrid
sel A  IDs B  max  IDs C  max  IDs D  max

result

late TR

max(B), max(C)
max(D), max(E)

result

hybrid
sel A | IDs B | max | IDs C | max | IDs D | max

Result:

late TR

hybrid

sel A | IDs B | C | D | E

max(B), max(C)
max(D), max(E)

Result:

hybrid
sel $A$ IDs $B$ max IDs $C$ max IDs $D$ max

result $0000$

late TR

max($B$), max($C$) max($D$), max($E$)

result

hybrid
sel A  IDs B  max  IDs C  max  IDs D  max

result

late TR

sel A  IDs B  C  D  E

max(B), max(C)
max(D), max(E)

result

hybrid
sel A IDs B max IDs C max IDs D max

result

late TR

sel A IDs B C D E

max(B), max(C) max(D), max(E)

result

hybrid
sel A → IDs B → max ID C → max ID D → max

result

late TR

sel A →

IDs B C D E
max(B), max(C)
max(D), max(E)

result

hybrid
sel A IDs B max IDs C max IDs D max

result 0000

late TR

sel A IDs B C D E

max(B), max(C) max(D), max(E)

result

hybrid
late TR

hybrid
late TR

hybrid
late TR

hybrid
The diagram illustrates the process of combining IDs from different regions. The top section shows the combination of IDs from regions A, B, C, D, and E, resulting in a hybrid set. The bottom section similarly combines IDs from regions A, B, C, D, and E to form a hybrid result. The process is represented by arrows indicating the flow of IDs and the addition of pairs (e.g., (r1, r2)).
late TR

hybrid
late TR

hybrid
late TR

hybrid
default
late tuple reconstruction (rather safe choice)

open topic for optimization
when and how to do selective early reconstruction

issues to consider
transformation overhead
materialization overhead
extra passes over the data
it may be almost for free (sometimes in hash joins)
dynamic code generation to fit data layouts

Read: DSM vs. NSM: CPU performance tradeoffs in block-oriented query processing
Marcin Zukowski, Niels Nes, Peter A. Boncz
International Workshop on Data Management on New Hardware (DaMoN) 2008
research papers

Browse: **Column-stores vs. row-stores: how different are they really?**
D. Abadi, S. Madden, and N. Hachem

Browse: **Positional update handling in column stores**
Sándor Héman, Marcin Zukowski, Niels J. Nes, Lefteris Sidirourgos, Peter A. Boncz

Browse: **Updating a cracked database**
Stratos Idreos, Martin Kersten, Stefan Manegold

Browse: **Integrating compression and execution in column-oriented database systems**
Daniel J. Abadi, Samuel Madden, Miguel Ferreira
Notes to remember

Column-stores and row-store are only extreme designs

Tuple reconstruction is key

and can happen anytime through a query plan

New opportunities for lightweight compression

given CPU/memory performance properties
more about
column-store plans
and compression

DATA SYSTEMS

prof. Stratos Idreos