column-stores basics 2.0
& data structures
prof. Stratos Idreos

HTTP://DASLAB.SEAS.HARVARD.EDU/CLASSES/CS265/
today:

papers & presentations

finish basics of modern db architectures

basics of data structures

systems project
first 12 paper slots are online (schedule tentative)
pick a paper slot: sign up as groups of 2 in google doc
read carefully both primary (P) and (B) papers
everyone: write a 1 page review for 5 papers
sign up for 5 papers in google doc
send PDF before class to TFs

come to OH often for help with reviews and slides

review and slides should focus on
what is the problem
why is it important
why is it hard
why existing solutions do not work
what is the core intuition for the solution
solution step by step
does the paper prove its claims
exact setup of analysis/experiments
are there any gaps in the logic/proof
possible next steps

* follow a few citations to gain more background
quick recap
virtual ids/ positional alignment

columns do not need to have the same width

positional lookups/joins

\[ A(i) = A + i \times \text{width}(A) \]
ok so now we can selectively read columns but how do we process them?

option 1

option 2

early tuple reconstruction/materialization
late reconstruction/materialization

```
select min(C) from R where A<10 & B<20
```

always sequential access patterns

memory contains only what is needed at any point in time
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
basic column-store details 2.0
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) $\text{select max}(B), \text{max}(C), \text{max}(D), \text{max}(E) \text{ from } R \text{ where } A>v_1$

2) $\text{select } B+C+D+E \text{ from } R \text{ where } A>v_1$

Should we use late or early tuple reconstruction plans?
sel A → IDs B → max → result

late TR

case 1:

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

hybrid
initial state columns in insertion order

sorted A B C

propagate order of A

sorted A B C
**select** max(D), min(E) **from** R **where** (A>10 and A<40) and (B>20 and B<60)

**avoid scan** of A
**avoid TR** on B
work on a **restricted area**
across all columns

good for memory hierarchy

- binary search for 10 & 40
- for all B values between pos1 & 2: if B>20 and B<60 mark bit vector at pos i
- for each marked position max(D)
\textbf{select} \text{max}(D), \text{min}(E) \textbf{from} R \textbf{where} (A>10 \text{ and } A<40) \textbf{or} (B>20 \text{ and } B<60)

\begin{itemize}
  \item \textbf{sorted}
  \item \textbf{binary search} for 10 & 40
  \item for all B values \textbf{outside} pos 1 & 2: if B>20 & B<60
  \item mark bit vector at pos i
  \item for each marked position max(D)
\end{itemize}
queries that filter on A benefit

queries that filter on B benefit
update \texttt{row7=(A=a,B=b,C=c,D=d)}

row-store vs column-store

which is better to update and why?
how much does it cost to update a single row?
(think about pages, data movement)
how to update in column-stores?
(query plan + algorithms)
query

base data

update

periodically

pending updates
A \hspace{0.5cm} B \hspace{0.5cm} C \hspace{0.5cm} D

query

A \hspace{0.5cm} B \hspace{0.5cm} C \hspace{0.5cm} D

optimizer

columns copy

rows copy

fractured mirrors
main-memory systems
optimized for the memory wall

with or without persistent data
other system categories
noSQL, new SQL, key-value stores, matlab, etc..

column-stores = bad name  modern systems
basics of data structures
index knows order about the data

filtering data: point/range queries
ok and how do we build, search, update a tree index efficiently?

structure = complexity = if statements, random access, instruction misses, etc. = no free lunch

node size, data organization fanout ...
searching internal node

(v1,p1) (v2,p2) (v3,p3) (v4,p4) (v5,p5) (v6,p6)…
searching leaf nodes

what does a leaf contain

searching leaf nodes
info to navigate lower level value-pointer

page size: 64K - holds 16K 4 byte ints
N elements, P pages

sorted array

1,2,3… 12,15,17 20,… 30,50 35,… 50,…

<12  >=12

can **index 8K pages** of the next level

4+4 bytes for each page (value+pointer)
64K/8= **index 8K pages**
A balanced search tree with random accesses to \( \log_{\text{fanout}} N \) height. Nodes represent intervals of values, with fanout connecting internal nodes to leaves. Leaves are labeled with sequences of random values. The diagram illustrates the hierarchical structure of the tree, with the root node at the top and leaves at the bottom. The height of the tree is logarithmic in the number of nodes, \( N \).
leaves

get 15
get 15-25

30,50
12,20 35,... 50,...
<12  >=12
1,2,3... 12,15,17 20,... 50,... 75,...
updates

1,2,3... 12,15,17 20,...

2,5,6,3,2 22,25,24

7,8,3,5,4

good
projects

systems development project is online
Modern B-Tree Techniques
by Goetz Graefe,
Foundations and Trends in Databases, 2011

research
http://daslab.seas.harvard.edu/rum-conjecture/
http://daslab.seas.harvard.edu/evolution/
buffer

L1

L2

any problems?
project definition and expectations are online

basics:
build LSM key-value store
put and get values (point queries)
scale with multi-cores
make it efficient >>1 reads/writes per second

evaluation: be able to demonstrate basic functionality and efficiency
you should have scripts for everything!
in person evaluation/demo + report with design and analysis
**midway check-in point in mid March**

we should be able to independently run your scripts/code with
minimum effort to verify your results:
work on a Linux-based environment (use a VM) and recent gcc

join often for OH to get hands on help with design and code
column-stores basics 2.0
& data structures

BIG DATA SYSTEMS

prof. Stratos Idreos