column-stores basics

prof. Stratos Idreos

HTTP://DASLAB.SEAS.HARVARD.EDU/CLASSES/CS265/
quick recap
- **CPU**
- **registers**
- **on chip cache**
- **on board cache**
- **SRAM** (~1ns)
- **DRAM** (~10ns)
- **memory** (~100ns)
- **disk**

**Memory Wall**

**Cache Miss**: looking for something which is not in the cache

**Memory Miss**: looking for something which is not in memory

- **Faster**
- **Cheaper**
Jim Gray, IBM, Tandem, DEC, Microsoft
ACM Turing award
ACM SIGMOD  Edgar F. Codd Innovations award
design

- logical design
- physical design
- system design
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/

project discussion on Friday+OH
today: basics of column+stores

next time: complex data structures & multi-cores
\textbf{select} \textit{min(A)} \textbf{from} \textit{R where} \textit{B<10 and C<80}
random access & page-based access

data value x

need to only read x… but have to read all of page 1

page1  page2  page3  …

data move

CPU
registers
on chip cache
on board cache
memory
disk
os block size

device block size

dbms block size

os and db will typically refer to **pages**
employee
(id:int, name:varchar(50), office:char(5), telephone:char(10), city:varchar(30), salary:int)

(1, name1, office1, tel1, city1, salary1)
(2, name2, office2, tel2, city2, salary2)
(3, name3, office3, tel3, city3, salary3)
(4, name4, office4, tel4, city4, salary4)
(5, name5, office5, tel5, city5, salary5)
(6, name6, office6, tel6, city6, salary6)
(7, name7, office7, tel7, city7, salary7)
(8, name8, office8, tel8, city8, salary8)
(9, name9, office9, tel9, city9, salary9)
...

data storage
blocks < pages < files

remember: the way we store data defines the best possible way we can access it
employee
(id:int, name:varchar(50), office:char(5), telephone:char(10), city:varchar(30), salary:int)

(1, name1, office1, tel1, city1, salary1)
(2, name2, office2, tel2, city2, salary2)
(3, name3, office3, tel3, city3, salary3)
(4, name4, office4, tel4, city4, salary4)
(5, name5, office5, tel5, city5, salary5)
(6, name6, office6, tel6, city6, salary6)
(7, name7, office7, tel7, city7, salary7)
(8, name8, office8, tel8, city8, salary8)
(9, name9, office9, tel9, city9, salary9)

...
slotted page

```
free_offset, N, offset1-length1, offset2-length2,…
```

---

```
update
```

```
var length
```

...
row-store

A B C D

file

stored continuously

one page contains all fields of multiple attributes

select A, B, C, D

select A
row-store

column-store

stored continuously

one page contains fields of a single attribute

select A, B, C, D

select A
the way we store data defines the possible (efficient) access methods
~1960s

1970: column storage ideas start appearing

~2000: open source complete system

1985: first rather complete column-store model

2005-now: more ideas and industry adoption of column-store designs

c-store, vertica, vectorwise, and then ibm, microsoft, oracle, and more
virtual ids/positional alignment

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?

A B C D

option 1

disk

memory

A

column-store engine

option 2

A B C

row-store engine

early tuple reconstruction/materialization
late reconstruction/materialization

```sql
SELECT min(C) FROM R WHERE A < 10 AND B < 20
```

always sequential access patterns
memory contains only what is needed at any point in time
working over fixed width & dense columns

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

**fetch**
for (i=0; i<size; i++)
  inter2[j++] = column[inter1[i]]

no function calls, no indirections, no auxiliary data, min ifs
easy to prefetch next data values
alt1) start with B
alt2) scan A & B independently and merge
alt3) store intermediates as bit vectors - not positions
…
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
possible data flow patterns

tuple at a time
block/vector at a time
column at a time
select min(C) from R where A<10 & B<20
column-stores basics

BIG DATA SYSTEMS

prof. Stratos Idreos