column-stores basics

prof. Stratos Idreos

HTTP://DASLAB.SEAS.HARVARD.EDU/CLASSES/CS265/
Goetz Graefe
Google Research
Justin Levandoski
Microsoft Research
option 1: systems project (now online)
basic key-value store functionality - work individually
single machine - multi-core design
basic design as in Facebook, LinkedIn, Mongo, etc.
can lead to research

option 2: research project
self-designing data systems + shape-shifting access methods
research with DASlab researchers - groups of 3
available for cs165 students or otherwise advanced students
next generation adaptive Key-value stores (with Facebook)
C/C++

no libraries unless we explicitly allow it
we expect you build everything from scratch
so you can control storage and access 100%

Basic Readings for Projects

Modern B-Tree Techniques
by Goetz Graefe,
Foundations and Trends in Databases, 2011

The Log-Structured Merge-Tree (LSM-Tree)
by Patrick E. O'Neil, Edward Cheng, Dieter Gawlick, Elizabeth J. O'Neil
midway check-in (10%)

special class (2-3 hour?) in mid March:
1) design docs
2) at least one performance example
3) presentation/poster
workload?

two reading sessions and two hacking sessions per week? so maybe **a minimum of 15 hours** per week (if you already have decent hacking and data structure/algorithms experience)
**Action steps:**
1) Read the syllabus & website carefully,
2) Register to Piazza,
3) Do P0 if you have not taken CS165 and check self-test,
4) Register for paper presentation (week 2),
5) Start submitting your paper reviews (week 3)

**web site:** http://daslab.seas.harvard.edu/classes/cs265/
**piazza:** piazza.com/harvard/spring2017/cs265/home
**office hours:** Stratos: Wed/Thur/Fri, 3-4pm, MD139
**TF office hours:** Mon 4-5pm, Tue, 3-4pm, MD 136
**textbook:** nope

research papers will be available from the Harvard network
how can I prepare?

1) start browsing some basic texts

**Get familiar with the very basics of traditional database architectures:**

**Get familiar with very basics of modern database architectures:**

**Get familiar with the very basics of modern large scale systems:**

2) play with basic data structures
implementation in C (linked list/hash table/tree)
class attendance != participating in discussion

**expectation**: on average everyone “tries” to speak 3-4 times
do not obsess about correctness, derailing class, etc.
design

logical design

physical design

system design
essential steps in using a database system

clean -> schema -> load -> tune

query

experts/system admins

user/apps
declarative interface
ask what you want

so do db systems “just work”?
declarative interface
ask what you want

indexes/views/tuning knobs

db system

DBA
declarative interface
ask what you want

\[ \downarrow \quad \uparrow \]

indexes/views/tuning knobs

but \ldots \ db cracking, adaptive* ideas

db system
design

- **logical** design
- **physical** design
- **system** design
select \( \min(A) \) from \( R \) where \( B<10 \) and \( C<80 \)
build a key-value store
similar to the ones Facebook, Google, etc use

interface supported: put, get, scan, count, get range, load
unique key-value pairs, r>>w but w>>0

data

how to store and access
- **CPU**
- **registers**
- **on chip cache**
- **on board cache**
- **memory**
- **disk**

**cache miss**: looking for something which is not in the cache

- **SRAM**: ~1ns
- **DRAM**: ~10ns
- **memory**: ~100ns

**memory miss**: looking for something which is not in memory

**faster**

**cheaper**

**speed**

**mem**

**time**

**memory wall**

CS265, Spring 2017
Stratos Idreos
Jim Gray, IBM, Tandem, DEC, Microsoft
ACM Turing award
ACM SIGMOD Edgar F. Codd Innovations award
random access & page-based access

need to only read $x$... but have to read all of page 1
os block size

device block size

os and db will typically refer to pages

dbms block size
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)
...

page

header

row1

row2

row3

...
slotted page

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

free space

scan
null
update
var length
…
row-store

select A, B, C, D

select A

one page contains all fields of multiple attributes

stored continuously

file
Row-store:

- Stored continuously

Column-store:

- 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

1985: first rather complete column-store model

~2000: open source complete system

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

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**: Early tuple reconstruction/materialization
  - Read columns from disk
  - Store in memory
  - Process in column-store engine

- **Option 2**: Row-store engine
  - Read whole row from disk
  - Store in memory
  - Process in row-store engine
late reconstruction/materialization

\[ \text{select } \min(C) \text{ from } R \text{ where } A < 10 \text{ & } B < 20 \]
late reconstruction/materialization

```
select min(C) from R where A<10 & B<20
```
late reconstruction/materialization

\textbf{select} \textbf{min}(C) \textbf{from} R \textbf{where} A<10 \& B<20

\begin{verbatim}
1:  int *input=A
2:  for (i=0;i<tuples;i++,input++)
3:    if *input<10
4:      *output=i
5:    output++
\end{verbatim}
late reconstruction/materialization

```
select min(C) from R where A<10 & B<20
```
late reconstruction/materialization

\[ \text{select } \min(C) \ \text{from} \ R \ \text{where} \ A < 10 \ & \ B < 20 \]
late reconstruction/materialization

```
select min(C) from R where A<10 & B<20
```
late reconstruction/materialization

```
select min(C) from R where A<10 & B<20
```
late reconstruction/materialization

\[ \text{select min}(C) \text{ from } R \text{ where } A<10 \text{ and } B<20 \]
late reconstruction/materialization

```sql
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
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]]
```
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
\begin{equation}
\text{select } \min(C) \text{ from } R \text{ where } A < 10 \& B < 20
\end{equation}
essential column-stores features
virtual ids
late tuple reconstruction (if ever)
vectorized execution
compression
fixed-width columns

Column-stores vs. row-stores: how different are they really?
D. Abadi, S. Madden, and N. Hachem
ACM SIGMOD Conference on Management of Data, 2008
but why now…
weren’t all those design options obvious in the past as well?

- moving data from disk
- moving data from memory
- computation

1) big memories
2) cpu vs memory speed
column-stores basics

BIG DATA SYSTEMS

prof. Stratos Idreos