logistics

Register for papers (3-4 papers will be added by the end of the week)

Systems project start (3 weeks to midway check-in)

Reviews start next week: Data Canopy for Monday and then readings as of W

Feedback on reviews: OH. We will grade but not count the first 6 reviews.

Announcements in class & piazza about logistics (but no worries otherwise)
Design Primitives

- fanout
- filter bits
- buffer size
- merge policy

Data structure designs are derived as combinations of fundamental design primitives.

Design Continuum

Data structures

Design Space
**fundamental** building blocks

**properties** when combined
fundamental building blocks
properties when combined

NoSQL

b-tree

Ism-tree

log+index
fundamental building blocks

properties when combined

NoSQL

find near instantly transition easily

b-tree

Ism-tree

log+index
PARETO OPTIMAL
BIJECTIVE
DIVERSE
NAVIGABLE
LAYERED
transitions from B-trees to LSM-trees and back
@SIGMOD 2019 URC
transitions

from B-trees to LSM-trees and back
@SIGMOD 2019 URC
transitions from B-trees to LSM-trees and back
@SIGMOD 2019 URC
that both

timum. The net result is that design continuums can be
represents a global minimum found experimentally. Tests
visited point. We then plot an orange dot. The yellow dot
until we either reach the edge of the simplex or a previously
Finally, for each grid location, the process follows the arrows
move 8 bytes from one component to the other every time).

top of the disk access contour plot pointing from the lowest
across all bloom filters. We evaluate all three gradients at

Figure 11 shows an example of our results for a skewed work-
highly sensitive to the workload. However, we can use the
sign (specifically the main memory bu
other memory allocations that uniquely characterize a de-
resources to allocate to the cache. What is hard about this
e, the bloom filters,

PointLookup (searchKey)
1. if $M_B > E$ then
2.     entry := buffer.find(searchKey);
3.   if entry then
4.     return entry;
5. // Pointer for direct block access. Set to root.
6. blockToCheck := levels[0].runs[0].nodes[0];
7. for i ← 0 to L do
8.   // Check each level's runs from recent to oldest.
9.   for j ← 0 to levels[i].runs.count do
10.   /* Prune search using bloom filters and fences
11.     when available. */
12.     if i < (L - Y) // At hot levels.
13.     then
14.       keyCouldExist :=
15.         filters[i][j].checkExists(searchKey);
16.     if !keyCouldExist then
17.       continue;
18.     else
19.       blockToCheck :=
20.         fences[i][j].find(searchKey);
21.     /* For oldest hot run, and all cold runs, if no
22.        entry is returned, then the search continues
23.        using a pointer into the next oldest run. */
24.     entry, blockToCheck :=
25.         blockToCheck.find(searchKey);
26.     if entry then
27.       return entry;
28. return keyDoesNotExist;

transitions

from B-trees to LSM-trees and back

@SIGMOD 2019 URC
WE NEED TO GO BEYOND ADAPTIVE STORAGE
WE NEED TO GO BEYOND ADAPTIVE STORAGE
A column-store that adaptively and incrementally creates indexes and loads data … is still a column-store
WE NEED TO GO BEYOND ADAPTIVE STORAGE

- System Design
- Prototype Implementation
- Full Implementation
- Set-up & Tune
- Auto-tuning during query processing

- Workloads, h/w expected properties (performance, throughput, energy, budget, ...)

Roles: Architects & Developers

Roles: Database Administrators

Design & Development

Set-up and Tuning

Application specific workload, h/w & expected properties
WE NEED TO GO BEYOND ADAPTIVE STORAGE

Design & Development
Roles: Architects & Developers

System Design
1

Prototype Implementation
2

Full Implementation
3

Set-up & Tune
4

Set-up and Tuning
Roles: Database Administrators

Auto-tuning during query processing
5

Roles: Database Administrators

application specific workload, h/w & expected properties

workloads, h/w expected properties (performance, throughput, energy, budget, …)

Design & Development
Roles: Architects & Developers

System Design
1

Prototype Implementation
2

Full Implementation
3

Set-up & Tune
4

Set-up and Tuning
Roles: Database Administrators

Auto-tuning during query processing
5

Roles: Database Administrators

application specific workload, h/w & expected properties

workloads, h/w expected properties (performance, throughput, energy, budget, …)
WE NEED TO GO BEYOND ADAPTIVE STORAGE

**System Design**
- Roles: Architects & Developers

1. System Design
2. Prototype Implementation
3. Full Implementation

**Design & Development**

- workloads, h/w expected properties (performance, throughput, energy, budget, …)

**Set-up and Tuning**
- Roles: Database Administrators

4. Set-up & Tune
5. Auto-tuning during query processing

**Application Specific**
- workload, h/w & expected properties
HOW CAN WE EASILY NAVIGATE THE WHOLE DESIGN SPACE?
assisted discovery
single result lookup cost (I/O)

- **O(1)**
- **O(log_B(N))**
- **O(B*log_B(N))**
- **O(1 + B*e^M_{bf}/N)**

**B+-tree**

**LSM-tree**

**B^*-tree**

**new designs**

**update cost (I/O)**

**data**

**index**

**assisted discovery**
**Assisted Discovery**

- **Data**: Bε-Tree with no order in levels
- **Data**: Tiered LSM with fractional cascading

**Update Cost (I/O)**

- **O(1)**
- **O(\log_B(N/B-D))**
- **O(1 + \log_B(N/B-D))**
- **O(\log_B(N))**
- **O(1)**

**Single Result Lookup Cost (I/O)**

- **O(1)**
- **O(\log_B(N))**
- **O(1 + \log_B(N))**
- **O(\log_B(N/B-D))**
- **O(\frac{1}{B} \log_B(N))**
- **O(\frac{1}{B} \log_B(N/B-D))**
- **O(\frac{1}{B} \log_B(N))**

**Index**: Data

**Tiered LSM** with fractional cascading
BεTree with no order in levels
Tiered LSM with fractional cascading

BεTree* hot levels
LSM-Tree* cold levels

data

index

assisted discovery
What is a new data structure?
substantially different/new performance properties

Is there a workload for every structure?
no idea…
IS THIS GOING TO REPLACE RESEARCHERS/ENGINEERS?

Did the arithmetic calculator replace mathematicians?
// Copyright (c) 2017 Harvard DASlab
// Simple bounds-checking sorted vector implementation
name "Vector"
description "A simple bounds-checking sorted vector implementation"

// Type definitions
key Int64
value Int64

// Data structure definition
size(Int64)

sort (bool)
Stores elements in sorted order
- bool - enable/disable
cost synthesis across the memory hierarchy (emerging hardware)

graphs, spatial data, images

cloud costs
### Concurrency Control

**Optimistic**: Operations are made without taking locks with validity checked after the operation and repeated if necessary. **Pessimistic**: Operations are blocked during modification but always succeed when executed. **Lockfree**: operations are made without locks using atomic operations.

#### Lock Type
- **Spin**
- **Yield[(Try)]**
- **Read-Write[(Try)]**

#### Granularity
- **Global**
- **Hand-over-hand**
- **Node**
- **Partition**
- **Key**
- **Byte**

#### Delete Method
- **Invalidate**
- **Removal**

#### Data Type
- **Values**
- **Keys**

#### Data Location
- **In-place**
- **Out-of-Place**

#### Modification Type
- **Delta**
- **Full**

#### Modification Timing
- **Eagerly**
- **Lazily**

#### Long Key Support
- **True**
- **False**

#### Value Compression
- **True(TYPE)**
- **False(TYPE)**

**Examples**

<table>
<thead>
<tr>
<th>Domain</th>
<th>Massertree</th>
<th>FASTER</th>
<th>BW-Tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Ops</td>
<td>SMO</td>
<td>Data Ops</td>
<td>SMO</td>
</tr>
<tr>
<td>R</td>
<td>U</td>
<td>I</td>
<td>D</td>
</tr>
<tr>
<td>Locking</td>
<td>Concurrency Control</td>
<td>Optimistic</td>
<td>Pessimistic</td>
</tr>
<tr>
<td>Data</td>
<td>Delete Method</td>
<td>Invalidate</td>
<td>Removal</td>
</tr>
<tr>
<td></td>
<td>Modification Location</td>
<td>In-place</td>
<td>Out-of-Place</td>
</tr>
<tr>
<td></td>
<td>Modification Type</td>
<td>Delta</td>
<td>Full</td>
</tr>
<tr>
<td></td>
<td>Modification Timing</td>
<td>Eagerly</td>
<td>Lazily</td>
</tr>
<tr>
<td></td>
<td>Long Key Support</td>
<td>True[(Try)]</td>
<td>False</td>
</tr>
<tr>
<td></td>
<td>Value Location</td>
<td>Embedded</td>
<td>Remote</td>
</tr>
<tr>
<td></td>
<td>Value Type</td>
<td>Fixed width</td>
<td>Variable</td>
</tr>
<tr>
<td></td>
<td>Key Compression</td>
<td>True(TYPE)</td>
<td>False</td>
</tr>
<tr>
<td></td>
<td>Value Compression</td>
<td>True(TYPE)</td>
<td>False</td>
</tr>
</tbody>
</table>
Leibniz, Let us calculate
Leibniz, Let us calculate
Can’t we do the same for other data-data-intensive problems?

statistics? neural networks?
How many and which structures are possible?

Can we compute performance w/o coding?
data science
always something new is going on

Algorithms | Systems | Analytic Pipelines
How to efficiently store and access data?  

Design custom systems tailored for these actions
How to efficiently store and access data?

Design custom systems tailored for these actions

**PROBLEM**: algorithms and processes are changing literally everyday…

**TAILORED SYSTEMS, HARDWARE OR EVEN EXTENSIONS ARE BOUND TO "FAIL"**
statistics are everywhere!
Temperature

May 2017
Temperature

May 2017
Correlation

- Lemonade Sale
- Temperature
- Hot Choc. Sale

Correlation
EXPLORATION & Repetitive Statistics
Repetition takes multiple forms
Repetition takes multiple forms
Repetition takes multiple forms

Different Statistics
Repetition takes multiple forms

Mixed

Query Range

Time

Q1
S1

Q2
S2

Q3
S3

Column
Repetition in real-life workloads

Repetition is everywhere - between 50% to 99%
How Do Existing Tools Perform?

NumPy (Python) | ModelTools (R) | MonetDB
How Do Existing Tools Perform?

NumPy (Python) | ModelTools (R) | MonetDB

Sequence of Queries

Q1 - Mean  Q2 - Var.  Q3 - Cov.  Q4 - Mean  Q5 - Var.  Q6 - Cov.
How Do Existing Tools Perform?

NumPy (Python) | ModelTools (R) | MonetDB

Sequence of Queries

Q1- Mean  Q2- Var.  Q3- Cov.  Q4- Mean  Q5- Var.  Q6- Cov.
How Do Existing Tools Perform?

| Q1- Mean | Q2- Var. | Q3- Cov. | Q4- Mean | Q5- Var. | Q6- Cov. |
How Do Existing Tools Perform?

- Q1 - Mean
- Q2 - Var.
- Q3 - Cov.
- Q4 - Mean
- Q5 - Var.
- Q6 - Cov.
How Do Existing Tools Perform?

Q1 - Mean  Q2 - Var.  Q3 - Cov.  Q4 - Mean  Q5 - Var.  Q6 - Cov.

NumPy
How Do Existing Tools Perform?

- NumPy

- Q1 - Mean
- Q2 - Var.
- Q3 - Cov.
- Q4 - Mean
- Q5 - Var.
- Q6 - Cov.
How Do Existing Tools Perform?

- NumPy
- Modeltools
- MonetDB

Q1- Mean  Q2- Var.  Q3- Cov.  Q4- Mean  Q5- Var.  Q6- Cov.
How Do Existing Tools Perform?

Q1 - Mean
Q2 - Var.
Q3 - Cov.
Q4 - Mean
Q5 - Var.
Q6 - Cov.

- NumPy
- Modeltools
- MonetDB
Data
Existing systems always compute statistics from scratch.
Data Canopy

Statistical queries

Library of building blocks

Avoid redundant data access to accelerate statistical analysis
Statistic
Statistic

Basic Aggregates

Data

a chunk
Q: Monthly Variance
Q: Monthly Variance

\[
\text{Variance} = \left(\frac{1}{n} \sum x^2\right) - \left(\frac{1}{n} \sum x\right)^2 \rightarrow \sum x \quad \sum x^2
\]
Q: Monthly Variance

\[
\text{Variance} = \left( \frac{1}{n} \sum x^2 \right) - \left( \frac{1}{n} \sum x \right)^2
\]
Q: Monthly Variance \[ \left( \frac{1}{n} \sum X^2 \right) - \left( \frac{1}{n} \sum X \right)^2 \]
**Var (first week)**  
\[ \frac{1}{7} \sum x_i^2 - \left( \frac{1}{7} \sum x_i \right)^2 \]  
Reuse between ranges

**Monthly mean**  
\[ \frac{1}{31} \sum x_i \]  
Reuse between statistics

**Mean (first week)**  
\[ \frac{1}{7} \sum x_i \]  
Mixed

Chunk size: 7 (a week)
\[
\text{Var (first week)} \quad \frac{1}{7} \sum x_i^2 - \left(\frac{1}{7} \sum x_i\right)^2
\]

Reuse between ranges

Monthly mean
\[
\frac{1}{31} \sum x_i
\]
Reuse between statistics

Mean (first week)
\[
\frac{1}{7} \sum x_i
\]
Mixed

Chunk size: 7 (a week)
Mean (first week) \[ \frac{1}{7} \sum x_i \] Mixed

Monthly mean \[ \frac{1}{31} \sum x_i \] Reuse between statistics

Var (first week) \[ \frac{1}{7} \sum x_i^2 - \left( \frac{1}{7} \sum x_i \right)^2 \] Reuse between ranges

Chunk size: 7 (a week)
Mean (first week) \[ \frac{1}{7} \sum x_i \]

Var (first week) \[ \frac{1}{7} \sum x_i^2 - \left( \frac{1}{7} \sum x_i \right)^2 \]

Monthly mean \[ \frac{1}{31} \sum x_i \]

Reuse between ranges

Reuse between statistics

Mixed

Chunk size: 7 (a week)
BIG DATA SYSTEMS

NoSQL | Neural Networks | SQL | Graph | Data Science