Systems Project is updated

Midway Check-in (March 10)
Design document/presentation/performance example

Start working ~Feb 20
We will go over the basic NoSQL design again
Discussion Papers
Automatic Database Management System Tuning Through Large-scale Machine Learning
KeystoneML: Optimizing Pipelines for Large-Scale Advanced Analytics
Fast Scans on Key-Value Stores
MISTIQUE: A System to Store and Query Model Intermediates for Model Diagnosis
The TileDB Array Data Storage Manager
Gist: Efficient Data Encoding for Deep Neural Network Training
ORPHEUSDB: Bolt-on Versioning for Relational Databases
LLAMA: Efficient graph analytics using Large Multiversioned Arrays

Discussion phase starts Feb 25
(or slightly later, depending on lecture progress)
(schedule will still move as we add (guest) lectures)
LSM-trees

- size ratio
- merge policy
- filters bits per entry
- size of buffer/cache
- internal k-v layout
LSM-trees
B-trees
Logs
Arrays
Bitmaps

- size ratio
- merge policy
- filters bits per entry
- size of buffer/cache
- key retention
- value retention
- partitioning
- sub-block links
- fanout
- internal k-v layout
- key retention
- value retention
- partitioning
- sub-block links
- fanout
key retention
value retention
partitioning
sub-block links
fanout
unified design space
POSSIBLE NODE DESIGNS
POSSIBLE NODE DESIGNS  POSSIBLE STRUCTURES
POSSIBLE STRUCTURES

- Trie
- Array
- Skip-List
- Hash-Table
- Linked-List
- Sorted Array

POSSIBLE NODE DESIGNS
POSSIBLE DATA LAYOUTS

read

write

operation

memory

read

write

memory

read

write

memory
ALGORITHM & COST SYNTHESIS
**POSSIBLE DATA LAYOUTS**

**read**

**write**

**memory**

**operation**

**RULES**

If ..., then ..., else

synthesize access pattern
sorted keys

columnar layout
sorted keys
columnar layout

RULES

sorted search
DEPENDS ON HARDWARE ENGINEERING

sorted keys columnar layout

RULES

sorted search

binary search1
binary search2
interpolation search1
interpolation search2
using new SIMD instruction X
...
COMPONENTS OF KEY-VALUE ALGORITHMS

RULES

- sorted keys
- columnar layout

- sorted search

- binary search1
- binary search2
- interpolation search1
- interpolation search2
- using new SIMD instruction X

- batched write
- BF probe
- scan

...
COMPONENTS OF KEY-VALUE ALGORITHMS

RULES

sorted keys
columnar layout

sorted search

batched write

BF probe

scan

binary search1
binary search2
interpolation search1
interpolation search2
using new SIMD instruction X

LEARNING

code, model

code, model

code, model

COMPONENTS OF KEY-VALUE ALGORITHMS
SYNTHESES FROM LEARNED MODELS

coding, modeling, generalized models, and a touch of ML

1. MINIMAL CODE

e.g., binary search

```cpp
if (data[middle] < search_val) {
    low = middle + 1;
} else {
    high = middle;
}
middle = (low + high)/2;
```

1 11 17 37 51 66 80 94
SYNTHESIS FROM LEARNED MODELS
coding, modeling, generalized models, and a touch of ML

1. MINIMAL CODE

```cpp
if (data[middle] < search_val) {
    low = middle + 1;
} else {
    high = middle;
}
middle = (low + high)/2;
```

e.g., binary search

2. BENCHMARK

![Graph showing time vs. data size in KB](image)
SYNTHESIS FROM LEARNED MODELS

coding, modeling, generalized models, and a touch of ML

1. MINIMAL CODE

```cpp
if (data[middle] < search_val) {
    low = middle + 1;
} else {
    high = middle;
}
middle = (low + high)/2;
```

e.g., binary search

2. BENCHMARK

![Graph showing Time (s) vs Data Size (KB)]

3. FIT MODEL

\[ f(x) = ax + b \log x + c \]
SYNTHESIS FROM LEARNED MODELS
coding, modeling, generalized models, and a touch of ML

1. MINIMAL CODE

- Example: binary search
  ```cpp
  if (data[middle] < search_val) {
    low = middle + 1;
  } else {
    high = middle;
  }
  middle = (low + high)/2;
  ```

2. BENCHMARK

- Run
- Graph showing time vs. data size

3. FIT MODEL

- Train
- Graph showing fitted model: $f(x) = ax + b \log x + c$

FOLDING ALGORITHMIC, ENGINEERING, AND H/W, PROPERTIES INTO THE COEFFICIENTS
FOR EACH OPERATION
FOR EACH OPERATION

1. Decide access **strategy** (L1) based on node design

2. Decide exact access strategy **implementation** (L2) based on available models

3. **Get cost** for chosen model
STATE GENERATION
LAYOUT SPEC & INSERTS

K fences-pointer pairs, sorted

T key-value pairs, no order

# of nodes & # entries in each node
computed cost = average cost
random access

**C++**

```cpp
for(int i=0; i<size; i++)
probes(array[pos[i]])
```

<table>
<thead>
<tr>
<th>pos</th>
<th>1</th>
<th>7</th>
<th>6</th>
<th>2</th>
<th>3</th>
<th>5</th>
<th>4</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>array</td>
<td>12</td>
<td>56</td>
<td>9</td>
<td>37</td>
<td>1</td>
<td>45</td>
<td>11</td>
<td>20</td>
</tr>
</tbody>
</table>

random/sequential access

**f(x)**

\[ f(x) = \sum_i \frac{c_i}{1 + e^{-k_i(x-x_i)}} \]

Run

Train

**Graph:**
- Time vs Region Size (KB)
- Memory

sum of sigmoids
Accessing Level 3

random access

C++
for(int i=0; i<size; i++)
    probe(array[pos[i]])

pos: 1 7 6 2 3 5 4 0
array: 1 2 5 6 9 3 7 1 4 5 1 1 1 2 0

Time (s) vs Region Size (KB)

\[ f(x) = \sum_{i} \frac{c_i}{1 + e^{-k_i(x - x_i)}} \]
Accessing Level 3

random access

\[
C++
\text{for}(\text{int } i=0; i<\text{size}; i++)
\text{ probe(array[pos[i]])}
\]

\[
\begin{array}{c}
\text{pos} \\
1 \ 7 \ 6 \ 2 \ 3 \ 5 \ 4 \ 0
\end{array}
\]

\[
\begin{array}{c}
\text{array} \\
12 \ 56 \ 9 \ 37 \ 1 \ 45 \ 11 \ 20
\end{array}
\]

\[
A f(x)
\]

\[
f(x) = \sum_{i} \frac{c_i}{1 + e^{-k_i(x-x_i)}}
\]

random/sequential access
EASY EXTENSIBILITY OF LEVEL 2 ACCESS PRIMITIVES

just adding a new benchmark for a Level 1 primitive
can be used in any design!
DESIGN SPACE

COST SYNTHESIS

HOW TO USE
what-if.design

CAN WE COMPUTE PERFORMANCE ACCURATELY?
what-if.design
CAN WE COMPUTE PERFORMANCE ACCURATELY?

layout spec → DC → cost

C++ → cost
what-if.design
CAN WE COMPUTE PERFORMANCE ACCURATELY?

layout spec → DC → cost

VS

C++ → cost

(same workload, hardware, data)
Response time (secs)

0.0000
0.0002
0.0004
0.0006
0.0008

Query Skew

0.5  1  1.5  2

CALCULATOR IMPLEMENTATION

B+Tree

{10M (uniform) k-v pairs, 100 point queries (skewed)}
Response time (secs)

B+Tree

CALCULATOR IMPLEMENTATION

Query Skew

0.5 1 1.5 2

{10M (uniform) k-v pairs, 100 point queries (skewed)}
Response time (secs)

0.0000
0.0002
0.0004
0.0006
0.0008

CALCULATOR
IMPLEMENTATION

B+Tree

Query Skew

0.5
1
1.5
2

Response time (secs)

{10M (uniform) k-v pairs, 100 point queries (skewed)}
The graph shows benchmark results for two different storage systems, `CALCULATOR` and `IMPLEMENTATION`, under varying query skew conditions.

- **B+Tree** graph: 
  - X-axis: Query Skew (0.5, 1, 1.5, 2) 
  - Y-axis: Response time (seconds) 
  - Observations: 
    - As query skew increases, the response time decreases for both `CALCULATOR` and `IMPLEMENTATION`. 
    - The `CALCULATOR` system consistently outperforms `IMPLEMENTATION` across all query skews.

- **CSB+Tree** graph: 
  - X-axis: Query Skew (0.5, 1, 1.5, 2) 
  - Y-axis: Response time (seconds) 
  - Observations: 
    - The performance trend is similar to `B+Tree`, with `CALCULATOR` consistently showing lower response times.

**Explanation:** 

- **Trial Details:** 
  - **Data Set:** 10M (uniform) k-v pairs 
  - **Queries:** 100 point queries (skewed)

The diagrams illustrate performance metrics under different query skew scenarios, emphasizing the efficiency of the `CALCULATOR` system over `IMPLEMENTATION` with regards to response time.
{10M (uniform) k-v pairs, 100 point queries (skewed)}
It works for numerous data structure classes and for diverse hardware and operations. Training cost 50-100 secs.
What-if we **add bloom filters** in the hash-table buckets?
What-if the workload changes to 90% writes?
What-if we *buy faster CPU X*?
What-if we buy faster CPU X?

~20 SECONDS
(workload: 10 Million entries, 100 queries)
AUTOMATICALLY REASON ABOUT performance and cost
How much faster does my system get if I spend another $50K per month?

I need to achieve response time X. What is the minimum I have to pay to Amazon Cloud?

I need to cut cloud expenses by $50K per month. What is the minimum performance loss?

AUTOMATICALLY REASON ABOUT performance and cost