CS 265
Stratos Idreos
BIG DATA SYSTEMS
NoSQL | Neural Networks | SQL | Graph | Data Science
Midway check in

Goal: Two working operations of basic design

Deliver: A design and performance analysis report

Schedule: Code review meeting

Week of March 9
POSSIBLE DATA LAYOUTS

read

write

memory

memory

memory

memory

memory

memory

memory

memory

DAI lab
© Harvard SEAS
POSSIBLE DATA LAYOUTS

operation

read

write

memory
COST SYNTHESIS ALGORITHM & POSSIBLE DATA LAYOUTS

ALGORITHM & COST SYNTHESIS
synthesize access pattern

write
read
memory

POSSIBLE DATA LAYOUTS

operation

RULES
If ..., then ..., else

synthesize access pattern
sorted keys
columnar layout
DEPENDS ON HARDWARE ENGINEERING

sorted keys columnar layout

RULES

sorted search

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

@ Harvard SEAS
COMPONENTS OF KEY-VALUE ALGORITHMS

RULES

- sorted keys
- columnar layout

- binary search 1
- binary search 2
- interpolation search 1
- interpolation search 2
- using new SIMD instruction X

COMPONENTS OF KEY-VALUE ALGORITHMS

sorted keys columnar layout

RULES

- sorted search
- ...
COMPONENTS OF KEY-VALUE ALGORITHMS

RULES

sorted keys
columnar layout

sorted search

batched write

BF probe

scan

LEARNING

binary search 1

binary search 2

interpolation search 1

interpolation search 2

using new SIMD instruction X

...
SYNTHESIS 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

e.g., binary search

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

2. BENCHMARK

[Graph showing time vs. data size]
SYNTHESIS FROM LEARNED MODELS
coding, modeling, generalized models, and a touch of ML

1. MINIMAL CODE

```cpp
C++

```e.g., binary search

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

2. BENCHMARK

![Graph showing time vs. data size](image)

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
e.g., binary search

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

2. BENCHMARK

Run

![Graph showing time vs. data size](data-graph)

3. FIT MODEL

Train

$f(x) = ax + b \log x + c$

FOLDING ALGORITHMIC, ENGINEERING, AND H/W, PROPERTIES INTO THE COEFFICIENTS
RULE/MODEL BASED SYSTEM SYNTHESIZES ALGORITHM AND COST

1. fanout.type = FIXED; 2. fanout.fixedVal = 20; 3. sorted = True; 4. zoneMaps.min = false; 5. zoneMaps.max = false; 6. retainData = false; 46. capacity = BALANCED;

Sorted search Random probe of zone-maps to fetch node

Sorted search Leaf data

Random probe to fetch leaf

Input

1. Internal Node
2. Leaf Node

Data Access Operation Synthesis Output

Cost Synthesis Output

Hardware Profile Data & Query Workload
TRAINING
TRAINING
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

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

T key-value pairs, no order
random access

C++

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

random/sequential access

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

random access

pos

1 7 6 2 3 5 4 0

array

12 56 9 37 1 45 11 20
Accessing Level 3

random access

C++

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

pos

1 7 6 2 3 5 4 0

array

12 56 9 37 1 45 11 20

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

random/sequential access
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

12 56 9 37 1 45 11 20

random/sequential access

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

Run

Train

sum of sigmoids

Time (s)

0.8e-8

L1, L2, L3

Memory

Region Size (KB)

8 32 1.664

Run

Train

sum of sigmoids

\[
f(x) = \sum_i \frac{c_i}{1 + e^{-k_i(x - x_i)}}
\]
EASY EXTENSIBILITY OF LEVEL 2 ACCESS PRIMITIVES

just adding a new benchmark for a Level 1 primitive

can be used in any design!
what-if.design

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

layout spec $\rightarrow$ DC $\rightarrow$ cost

C++ $\rightarrow$ cost
what-if.design
CAN WE COMPUTE PERFORMANCE ACCURATELY?

layout spec → DC → cost vs C++ → cost
(same workload, hardware, data)
Response time (secs)

<table>
<thead>
<tr>
<th>Query Skew</th>
<th>CALCULATOR</th>
<th>IMPLEMENTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

B+Tree

{10M (uniform) k-v pairs, 100 point queries (skewed)}
0.0000
0.0002
0.0004
0.0006
0.0008
0.00000
0.00004
0.00008
0.00012
0.00016
0.00020
0.00024
0.00028
0.00032
0.00036
0.00040
0.00044
0.00048
0.00052
0.00056
0.00060

CALCULATOR
IMPLEMENTATION
B+Tree

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

CALCULATOR
IMPLEMENTATION

B+Tree

Query Skew

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

<table>
<thead>
<tr>
<th>Query Skew</th>
<th>Calculator</th>
<th>Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>0.0008</td>
<td>0.0006</td>
</tr>
<tr>
<td>1</td>
<td>0.0008</td>
<td>0.0006</td>
</tr>
<tr>
<td>1.5</td>
<td>0.0008</td>
<td>0.0006</td>
</tr>
<tr>
<td>2</td>
<td>0.0008</td>
<td>0.0006</td>
</tr>
</tbody>
</table>

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

CALCULATOR vs IMPLEMENTATION

B+Tree

CSB+Tree

Query Skew

0.5  1  1.5  2

B

Tree

{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

var h/w and op

various data structures
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*?
~20 SECONDS
(workload: 10 Million entries, 100 queries)

What-if we **buy faster CPU X**?
Stratos Idreos, Konstantinos Zoumpatianos, Brian Hentschel, Michael Kester, Demi Guo. In Proceedings of the ACM SIGMOD International Conference on Management of Data, 2018

+ Technical Report if you want to see design primitives in more detail
Function $CompleteDesign\ (Q, \mathcal{E}, l, currentPath = [], H)$

1. if blockReachedMinimumSize($H\.page\.size$) then
   1. return END\_SEARCH;
2. if !meaningfulPath($currentPath, Q, l$) then
   1. return END\_SEARCH;
3. if (cacheHit = cachedSolution($Q, l, H$)) != null then
   1. return cacheHit;
4. bestSolution = initializeSolution(cost=$\infty$);
5. for $E \in \mathcal{E}$ do
6.     tmpSolution = initializeSolution();
7.     tmpSolution.cost = synthesizeGroupCost($E, Q$);
8.     updateCost($E, Q, \text{tmpSolution.cost}$);
9.     if createsSubBlocks($E$) then
10.        $Q' = create\_Query\_Blocks\ (Q)$;
11.        currentPath.append($E$);
12.        subSolution = CompleteDesign($Q', \mathcal{E}, l + 1, currentPath$);
13.        if subSolution.cost != END\_SEARCH then
14.           tmpSolution.append(subSolution);
15.        if tmpSolution.cost $\leq$ bestSolution.cost then
16.           bestSolution = tmpSolution;
17.    cacheSolution($Q, l, bestSolution$);
18. return bestSolution;