BIG DATA SYSTEMS

NoSQL | Neural Networks | SQL | Graph | Data Science
merging small levels does not help that much (point, range, space)
25% monthly savings $\$\$ vs Google LevelDB

Amazon Cloud (North America)
25% monthly savings $\$$ vs Google LevelDB

Amazon Cloud (North America)
Research
Everyone can “think” as long as they understand the fundamentals.

Solution: Read, read, read
Research
Everyone can “think” as long as they understand the fundamentals.

Solution: Read, read, read

Actually doing it
Engineering, Systems programming, Performance tuning
Understanding hardware, Hardware-conscious math modeling, Optimization (multi-variate calculus), Machine learning

Solution: work with Subarna, Hao, Utku, Sanket and Stratos on systems and research projects = learn from experience
Research
Everyone can “think” as long as they understand the fundamentals.

Solution: Read, read, read

Actually doing it
Engineering, Systems programming, Performance tuning
Understanding hardware, Hardware-conscious math modeling, Optimization (multi-variate calculus), Machine learning

Solution: work with Subarna, Hao, Utku, Sanket and Stratos on systems and research projects = learn from experience

Expectations: It takes time (and it never ends). Enjoy the process!
Today
System project: quick first intro
How to search the design space: Step 1: cost synthesis.
Systems Project

Design and build a full LSM-tree based key-value store
Scale up. Not out.
Systems Project

Design and build a full LSM-tree based key-value store
Scale up. Not out.

Why bother?

Most widely used type of system today across industries.
All challenges of today need to be solved within or on top LSMs.
Go deep into systems/performance engineering
What to do

Read the description in detail and start designing.
Attend the section next week.
**What to do**

Read the description in detail and start designing. Attend the section next week.

**Timeline**

Follow step by step timeline detailed in project description

Be ready for midway check-in
Basic design

Look carefully at: Basic requirements. We expect a few basic elements (filters, fence pointers, levels). But the design is purposely open:

   you are expected to make and explain several decisions.
Basic design

Look carefully at: Basic requirements. We expect a few basic elements (filters, fence pointers, levels). But the design is purposely open:

you are expected to make and explain several decisions.

Three Optimizations and Experiments

At least three ideas on how to optimize on basic design. Study, evaluate, present.
Sections for detailed intro to systems project

Feb 16, 2-3pm in person
Feb 18, 11am-noon Zoom
Sections for detailed intro to systems project

Feb 16, 2-3pm in person
Feb 18, 11am-noon Zoom

Can I switch to research project?
No. We have tried it. It does not work.
Research projects require systems expertise.

There are no shortcuts. Enjoy the ride!
Research is with us is always open for everyone once you get the basics in systems.
DESIGN SPACE

COST SYNTHESIS

WHAT-IF
STARS IN THE SKY

10^{24}

POSSIBLE DATA STRUCTURES

10^{32}, 2-node

10^{48}, 3-node
Memory

Read

Update

Memory
COST?
What would the performance be if we were to implement that design in a specific programming language and test a specific workload on a specific hardware?
What would the performance be if we were to implement that design in a specific programming language and test a specific workload on a specific hardware?

If we have the cost for 2 designs, we can compare them, and we can build search algorithms.
HOW TO JUDGE A DESIGN?
HOW TO JUDGE A DESIGN?

1

COMPLEXITY ANALYSIS
HOW TO JUDGE A DESIGN?

1. COMPLEXITY ANALYSIS
2. IMPLEMENTATION & TESTING
HOW TO JUDGE A DESIGN?

1. COMPLEXITY ANALYSIS
2. IMPLEMENTATION & TESTING
3. GENERALIZED MODELS
HOW TO JUDGE A DESIGN?

1. COMPLEXITY ANALYSIS
2. IMPLEMENTATION & TESTING
3. GENERALIZED MODELS

This sounds ideal: is it possible?
ACCESS PATH SELECTION in ANALYTICAL SYSTEMS
scan vs secondary index selection

data* system
ACCESS PATH SELECTION in ANALYTICAL SYSTEMS
scan vs secondary index selection

@SIGMOD 2017
ACCESS PATH SELECTION

scan vs secondary index selection

Pat Selinger
ACCESS PATH SELECTION
scan vs secondary index selection

P. Selinger et. al, 1979

Scan is best

Index is best

selectivity
ACCESS PATH SELECTION
scan vs secondary index selection

P. Selinger et. all, 1979

DO WE STILL NEED INDEXING? (AND IF YES HOW DO WE CHOOSE)
ACCESS PATH SELECTION in ANALYTICAL SYSTEMS
scan vs secondary index selection

P. Selinger et. all, 1979

Scan is best

Index is best

@SIGMOD 2017
ACCESS PATH SELECTION in ANALYTICAL SYSTEMS
scan vs secondary index selection

P. Selinger et all, 1979

Scan is best

Index is best

multi-core, SIMD, compression, columnar/hybrid, scan sharing, ...

# of concurrent queries
ACCESS PATH SELECTION in ANALYTICAL SYSTEMS
scan vs secondary index selection

P. Selinger et. al., 1979

selectivity
Index is best
Scan is best

multi-core, SIMD, compression, columnar/hybrid, scan sharing, …

# of concurrent queries
10%

Dawn of time
2000
2010
2017
Future

selectivity threshold
10%
1%
0%

Hardware Improvements
Column Stores
Main Memory
latency
bandwidth
scan vs secondary index selection @SIGMOD 2017

\[
APS(q, S_{tot}) = \frac{q \cdot 1 + [\log_b(N)] \cdot \left( BW_S \cdot C_M + b \cdot BW_S \cdot C_A + \frac{b \cdot BW_S \cdot f_p \cdot p}{2} \right)}{\max(ts, 2 \cdot f_p \cdot p \cdot q \cdot BW_S) + S_{tot} \cdot rw \cdot \frac{BW_S}{BW_R}} + \frac{S_{tot} \cdot \log_2 (S_{tot} \cdot N) \cdot BW_S \cdot C_A}{\max(ts, 2 \cdot f_p \cdot p \cdot q \cdot BW_S) + S_{tot} \cdot rw \cdot \frac{BW_S}{BW_R}} + \frac{S_{tot} \cdot log_2 (S_{tot} \cdot N) \cdot BW_S \cdot C_A}{\max(ts, 2 \cdot f_p \cdot p \cdot q \cdot BW_S) + S_{tot} \cdot rw \cdot \frac{BW_S}{BW_R}}
\]

### Workload

<table>
<thead>
<tr>
<th>Workload</th>
<th>$q$</th>
<th>number of queries</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_i$</td>
<td>selectivity of query $i$</td>
<td></td>
</tr>
<tr>
<td>$S_{tot}$</td>
<td>total selectivity of the workload</td>
<td></td>
</tr>
</tbody>
</table>

### Dataset

<table>
<thead>
<tr>
<th>Dataset</th>
<th>$N$</th>
<th>data size (tuples per column)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_s$</td>
<td>tuple size (bytes per tuple)</td>
<td></td>
</tr>
</tbody>
</table>

### Hardware

<table>
<thead>
<tr>
<th>Hardware</th>
<th>$C_A$</th>
<th>L1 cache access (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_M$</td>
<td>LLC miss: memory access (sec)</td>
<td></td>
</tr>
<tr>
<td>$BW_S$</td>
<td>scanning bandwidth (GB/s)</td>
<td></td>
</tr>
<tr>
<td>$BW_R$</td>
<td>result writing bandwidth (GB/s)</td>
<td></td>
</tr>
<tr>
<td>$B_W_l$</td>
<td>leaf traversal bandwidth (GB/s)</td>
<td></td>
</tr>
<tr>
<td>$p$</td>
<td>The inverse of CPU frequency</td>
<td></td>
</tr>
<tr>
<td>$f_p$</td>
<td>Factor accounting for pipelining</td>
<td></td>
</tr>
</tbody>
</table>

### Scan & Index

<table>
<thead>
<tr>
<th>Scan &amp; Index</th>
<th>$rw$</th>
<th>result width (bytes per output tuple)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b$</td>
<td>tree fanout</td>
<td></td>
</tr>
<tr>
<td>$aw$</td>
<td>attribute width (bytes of the indexed column)</td>
<td></td>
</tr>
<tr>
<td>$ow$</td>
<td>offset width (bytes of the index column offset)</td>
<td></td>
</tr>
</tbody>
</table>
scan vs secondary index selection @SIGMOD 2017

Tree Traversal + Leaf Traversal + Result Writing + Sorting

Base Scan + Predicate Eval. + Result Writing

\[ APS(q, S_{\text{tot}}) = \frac{q \cdot \left( 1+ \left\lfloor \log_b(N) \right\rfloor \right) \cdot \left( BW_S \cdot C_M + \frac{b \cdot BW_S \cdot C_A}{2} + \frac{b \cdot BW_S \cdot f_p \cdot p}{2} \right)}{\max \left( ts, 2 \cdot f_p \cdot p \cdot q \cdot BW_S \right) + S_{\text{tot}} \cdot rw \cdot \frac{BW_S}{BW_R}} \]

\[ S_{\text{tot}} \left( \frac{BW_S \cdot C_M}{b} + (aw + ow) \cdot \frac{BW_S}{BW_I} + rw \cdot \frac{BW_S}{BW_R} \right) + \]

\[ \max \left( ts, 2 \cdot f_p \cdot p \cdot q \cdot BW_S \right) + S_{\text{tot}} \cdot rw \cdot \frac{BW_S}{BW_R} \]

\[ S_{\text{tot}} \cdot \log_2 (S_{\text{tot}} \cdot N) \cdot BW_S \cdot C_A + \]

\[ \max \left( ts, 2 \cdot f_p \cdot p \cdot q \cdot BW_S \right) + S_{\text{tot}} \cdot rw \cdot \frac{BW_S}{BW_R} \]

<table>
<thead>
<tr>
<th>Workload</th>
<th>( q )</th>
<th>number of queries</th>
</tr>
</thead>
<tbody>
<tr>
<td>( s_i )</td>
<td>selectivity of query ( i )</td>
<td></td>
</tr>
<tr>
<td>( S_{\text{tot}} )</td>
<td>total selectivity of the workload</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dataset</th>
<th>( N )</th>
<th>data size (tuples per column)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_s )</td>
<td>tuple size (bytes per tuple)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Hardware</th>
<th>( C_A )</th>
<th>L1 cache access (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( C_M )</td>
<td>LLC miss: memory access (sec)</td>
<td></td>
</tr>
<tr>
<td>( BW_S )</td>
<td>scanning bandwidth (GB/s)</td>
<td></td>
</tr>
<tr>
<td>( BW_R )</td>
<td>result writing bandwidth (GB/s)</td>
<td></td>
</tr>
<tr>
<td>( B_W_I )</td>
<td>leaf traversal bandwidth (GB/s)</td>
<td></td>
</tr>
<tr>
<td>( p )</td>
<td>The inverse of CPU frequency</td>
<td></td>
</tr>
<tr>
<td>( f_p )</td>
<td>Factor accounting for pipelining</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Scan &amp; Index</th>
<th>( rw )</th>
<th>result width (bytes per output tuple)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( b )</td>
<td>tree fanout</td>
<td></td>
</tr>
<tr>
<td>( aw )</td>
<td>attribute width (bytes of the indexed column)</td>
<td></td>
</tr>
<tr>
<td>( ow )</td>
<td>offset width (bytes of the index column offset)</td>
<td></td>
</tr>
</tbody>
</table>
S. BING YAO
models/advisors

STEFAN MANEGOLD
model synthesis
We need something else: Something more scalable & robust!
POSSIBLE DATA LAYOUTS

read

write

memory

memory

memory

memory

memory

memory

memory

memory
POSSIBLE DATA LAYOUTS

read

write

memory

operation
ALGORITHM & COST SYNTHESIS
synthesize access pattern

POSSIBLE DATA LAYOUTS

If ..., then ..., else

operate

read

write

memory

rules

synthesize access pattern
sorted keys
columnar layout
sorted keys
columnar layout

RULES

sorted search
DEPENDS ON HARDWARE ENGINEERING

RULES

sorted keys columnar layout

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

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

batched write

BF probe

scan

LEARNING

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

code, model

code, model

code, model
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;
```
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
   e.g., binary search
   ```cpp
   if (data[middle] < search_val) {
     low = middle + 1;
   } else {
     high = middle;
   } middle = (low + high)/2;
   ```

2. BENCHMARK
   - Time vs. Data Size
   - Log-Linear Model
   - $f(x) = ax + b \log x + c$

3. FIT MODEL
   - Train
   - $f(x)$

DASlab
@ Harvard SEAS
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]

3. FIT MODEL

\[ f(x) = ax + b \log x + c \]

FOLDING ALGORITHMIC, ENGINEERING, AND H/W, PROPERTIES INTO THE COEFFICIENTS
<table>
<thead>
<tr>
<th>Data Access Primitives Level 1 (required parameters; optional parameters)</th>
<th>Model Parameters</th>
<th>Data Access Primitives Layer 2</th>
<th>Fitted Models</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Scan</strong> (Element Size, Comparison, Data Layout; None)</td>
<td>Data Size</td>
<td>Scalar Scan (RowStore, Equal)</td>
<td>Linear Model (1)</td>
</tr>
<tr>
<td><strong>Sorted Search</strong> (Element Size, Data Layout; )</td>
<td>Data Size</td>
<td>Binary Search (RowStore)</td>
<td>Log-Linear Model (2)</td>
</tr>
<tr>
<td><strong>Hash Probe</strong> (; Hash Family)</td>
<td>Structure Size</td>
<td>Linear Probing (Multiply-shift [29])</td>
<td>Sum of Sigmoid (5), Weighted Nearest Neighbors (7)</td>
</tr>
<tr>
<td><strong>Bloom Filter Probe</strong> (; Hash Family)</td>
<td>Structure Size, Number of Hash Functions</td>
<td>Bloom Filter Probe (Multiply-shift [29])</td>
<td>Sum of Sigmoid (6), Weighted Nearest Neighbors (7)</td>
</tr>
<tr>
<td><strong>Sort</strong> (Element Size; Algorithm)</td>
<td>Data Size</td>
<td>QuickSort</td>
<td>NLogN Model (4)</td>
</tr>
<tr>
<td><strong>Random Memory Access</strong></td>
<td>Region Size</td>
<td>Random Memory Access</td>
<td>Sum of Sigmoid (5), Weighted Nearest Neighbors (7)</td>
</tr>
<tr>
<td><strong>Batched Random Memory Access</strong></td>
<td>Region Size</td>
<td>Batched Random Memory Access</td>
<td>Sum of Sigmoid (5), Weighted Nearest Neighbors (7)</td>
</tr>
<tr>
<td><strong>Unordered Batch Write</strong> (Layout: )</td>
<td>Write Data Size</td>
<td>Contiguous Write (RowStore)</td>
<td>Linear Model (1)</td>
</tr>
<tr>
<td><strong>Ordered Batch Write</strong> (Layout: )</td>
<td>Write Data Size, Data Size</td>
<td>Batch Ordered Write (RowStore)</td>
<td>Linear Model (1)</td>
</tr>
<tr>
<td><strong>Scattered Batch Write</strong></td>
<td>Number of Elements, Region Size</td>
<td>ScatteredBatchWrite</td>
<td>Sum of Sigmoid (6), Weighted Nearest Neighbors (7)</td>
</tr>
</tbody>
</table>
RULE/MODEL BASED SYSTEM SYNTHESIZES ALGORITHM AND COST

**Per Node Access Operation Synthesis**

- **Serial Scan**
  - Zone Maps
  - Sorted = True
  - Fanout.type = FIXED

- **Sorted Search**
  - Zone Maps
  - Sorted = True
  - Fanout.type = FIXED

**Sub-block data distribution**

1. Create blocks using:
   - Partitioning property
   - Capacity property
   - Fanout property
2. Distribute data in blocks

**Operation Synthesis Output**

- Sorted search Random probe to fetch node
- Random probe to fetch leaf
- BinarySearch
  - BS 10, K
  - BS 20, K
  - BS 64, KV

**Internal Node**

1. Fanout.type = FIXED
2. Fanout.FixedVal = 64
3. Sorted = True
4. ZoneMaps.min = false
5. ZoneMaps.max = false
6. RetainsData = false
7. Capacity = BALANCED

**Leaf Node**

1. Fanout.type = FIXED
2. Fanout.FixedVal = 64
3. Sorted = True
4. ZoneMaps.min = false
5. ZoneMaps.max = false
6. RetainsData = false
7. Capacity = BALANCED

**Sorted search Random probe to fetch node**

- BS 10, K
- BS 20, K
- BS 64, KV

**Input**

- Structure Layout Specifications
- Hardware Profile
- Data & Query Workload

**Cost Synthesis Output**

- BinarySearch
- RandomProbe
TRAINING
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

T key-value pairs, no order

# of nodes & # entries in each node

computed cost = average cost
random access

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

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

random access

C++

```cpp
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

Run

Train

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

sum of sigmoids
Accessing Level 3

random access

C++

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

<table>
<thead>
<tr>
<th>pos</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</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>

run

Train

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

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!
CAN WE COMPUTE PERFORMANCE ACCURATELY?
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

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

Query Skew

B+Tree

CALCULATOR
IMPLEMENTATION

{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>0.0007</td>
</tr>
<tr>
<td>1.0</td>
<td>0.0005</td>
<td>0.0004</td>
</tr>
<tr>
<td>1.5</td>
<td>0.0003</td>
<td>0.0002</td>
</tr>
<tr>
<td>2.0</td>
<td>0.0002</td>
<td>0.0001</td>
</tr>
</tbody>
</table>

B+Tree

{10M (uniform) k-v pairs, 100 point queries (skewed)}
{10M (uniform) k-v pairs, 100 point queries (skewed)}
calculator
Implementation
CSB+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

<table>
<thead>
<tr>
<th>Linked-list</th>
<th>Array</th>
<th>Range Part. LL</th>
<th>Skip List</th>
<th>Trie</th>
<th>B+Tree</th>
<th>Sorted Array</th>
<th>Hash-table</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Graph" /></td>
<td><img src="image2.png" alt="Graph" /></td>
<td><img src="image3.png" alt="Graph" /></td>
<td><img src="image4.png" alt="Graph" /></td>
<td><img src="image5.png" alt="Graph" /></td>
<td><img src="image6.png" alt="Graph" /></td>
<td><img src="image7.png" alt="Graph" /></td>
<td><img src="image8.png" alt="Graph" /></td>
</tr>
</tbody>
</table>

var h/w and op

- HW1
  - point gets
  - CPU: 64x 2.3GHz
  - L3: 46MB
  - RAM: 256GB

- HW2
  - point gets
  - CPU: 4x2 3GHz
  - L3: 46MB
  - RAM: 16GB

- HW3
  - point gets
  - CPU: 64x2GHz
  - L3: 16MB
  - RAM: 1TB

- HW3
  - updates
  - CPU: 64x2GHz
  - L3: 16MB
  - RAM: 1TB

- HW3
  - range gets
  - CPU: 64x2GHz
  - L3: 16MB
  - RAM: 1TB
DESIGN SPACE | COST SYNTHESIS | HOW TO USE
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)
WE CAN AUTOMATICALLY DESIGN/DEBUG/FILL IN
WE CAN AUTOMATICALLY DESIGN/DEBUG/FILL IN
WE CAN AUTOMATICALLY DESIGN/DEBUG/FILL IN
Auto-tuning and Adaptive Systems. Typically, in these lines of work the layout adapts to incoming re-
incrementally building a speci-
spaces, typically to solve a very speci-
using feedback from tests. While there are shared concepts with 

**Algorithm 1: Complete a partial data structure layout specification.**

```plaintext
Function CompleteDesign (Q, E, l, currentPath = [], H)
    if blockReachedMinimumSize(H.page_size) then
        return END_SEARCH;
    if !meaningfulPath(currentPath, Q, l) then
        return END_SEARCH;
    if (cacheHit = cachedSolution(Q, l, H)) != null then
        return cacheHit;
    bestSolution = initializeSolution(cost=∞);
    for E ∈ E do
        tmpSolution = initializeSolution();
        tmpSolution.cost = synthesizeGroupCost(E, Q);
        updateCost(E, Q, tmpSolution.cost);
        if createsSubBlocks(E) then
            Q’ = createQueryBlocks(Q);
            currentPath.append(E);
            subSolution = CompleteDesign(Q’, E, l + 1, currentPath);
            if subSolution.cost != END_SEARCH then
                tmpSolution.append(subSolution);
        if tmpSolution.cost ≤ bestSolution.cost then
            bestSolution = tmpSolution;
    cacheSolution(Q, l, bestSolution);
    return bestSolution;
```

**A ADDITIONAL RELATED AREAS**

Current query marked as

Start

"unsorted" (U)

"sorted" (S)

"probe" (P)

actions on Database Systems (TODS)

of Data

Kostas Zoumpatianos, Stratos Idreos, and Themis Palpanas. 2014. Indexing for interactive

S. Bing Yao and D. DeJong. 1978. Evaluation of Database Access Paths. In 

PACMPL
Data Layout and Index Synthesis

Generalized Cost and Algorithm Synthesis

Data Access Primitives
- Serial Scan
- Equality Scan
- Range Scan
- Sorted Search
- Binary Search
- Random Probe

Hardware Profiles
- Equality Scan
- Range Scan
- Binary Search

Cost Synthesizer
- Machine Learning
  - Micro-benchmarks train models on different hardware profiles.
  - \( f(x) = ax + b \)

Operation Synthesis
- Translation
- Level 1 to Level 2 translation

Concurrency & Updates
- Bulk Load
- Put
- Get
- Delete
- Parallelization

Layout Primitives
- Data Node (Element) Library
- Internal Data Page
- B+ Tree
- B-Tree
- Trie
- Skip List
- Linked List
- Array

Overall Designs
- Design Space
- Design Continuums
- Node has partitioning?
- Node has Bloom filters?
- Node has Zone maps?

Design Continuums
- Fast analytical model optimization
- \( \arg \min f(x) \)
- Memoization
- H/W Pruning
- Expert Rules

Node by node design process
- Systematically evaluate various designs for each node
- Is it a design continuum?
- Otherwise

Design Search
- Reinforcement Learning
- Bayesian Optimization
- Genetic Algorithms

Learned Shortcuts
- Workload
- Hardware
- SLAs

Performance Constraints
- Initial design
- Time threshold
- Distance to optimal

Input
- High confidence
- Workload
- Hardware
- SLAs

Output
- Data structure design
- Code
- C++

Feedback
- Systematic evaluation
- Various designs
- Design Continuums

Machine Learning
- Micro-benchmarks train models on different hardware profiles.
- \( f(x) = ax + b \)

Operation Synthesis (Level 1)
- Hardware Conscious Synthesis (Level 2)

Performance Constraints
- Initial design
- Time threshold
- Distance to optimal

Data Node (Element) Library
- B-Tree Internal
- B-Tree
- Trie
- Skip List
- Linked List
- Array

Key Value Data Structures
- Design Space
- Data Layout and Index Synthesis

Currently solving
- Not yet solved
- Partial design

High confidence
- Workload
- Hardware
- SLAs

Low confidence
- Feedback

Update design
- Partial design
- Not yet solved
- Partial design

Learned Shortcuts
- Workload
- Hardware
- SLAs

Concurrent & Updates
- Bulk Load
- Put
- Get
- Delete

Update design
- Partial design
- Not yet solved
- Partial design

Learned Shortcuts
- Workload
- Hardware
- SLAs

Concurrent & Updates
- Bulk Load
- Put
- Get
- Delete

Learned Shortcuts
- Workload
- Hardware
- SLAs

Concurrent & Updates
- Bulk Load
- Put
- Get
- Delete

Learned Shortcuts
- Workload
- Hardware
- SLAs

Concurrent & Updates
- Bulk Load
- Put
- Get
- Delete

Learned Shortcuts
- Workload
- Hardware
- SLAs

Concurrent & Updates
- Bulk Load
- Put
- Get
- Delete

Learned Shortcuts
- Workload
- Hardware
- SLAs

Concurrent & Updates
- Bulk Load
- Put
- Get
- Delete

Learned Shortcuts
- Workload
- Hardware
- SLAs

Concurrent & Updates
- Bulk Load
- Put
- Get
- Delete

Learned Shortcuts
- Workload
- Hardware
- SLAs

Concurrent & Updates
- Bulk Load
- Put
- Get
- Delete

Learned Shortcuts
- Workload
- Hardware
- SLAs

Concurrent & Updates
- Bulk Load
- Put
- Get
- Delete

Learned Shortcuts
- Workload
- Hardware
- SLAs

Concurrent & Updates
- Bulk Load
- Put
- Get
- Delete

Learned Shortcuts
- Workload
- Hardware
- SLAs

Concurrent & Updates
- Bulk Load
- Put
- Get
- Delete

Learned Shortcuts
- Workload
- Hardware
- SLAs

Concurrent & Updates
- Bulk Load
- Put
- Get
- Delete

Learned Shortcuts
- Workload
- Hardware
- SLAs

Concurrent & Updates
- Bulk Load
- Put
- Get
- Delete
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
BIG DATA SYSTEMS

NoSQL | Neural Networks | SQL | Graph | Data Science