CS 265
BIG DATA SYSTEMS
Stratos Idreos
NoSQL | Neural Networks | SQL | Graph | Data Science
Presentations/paper discussions start on March 2

slides should be reviewed together with Subarna a week before

then a couple of days before with Stratos
Presentations/paper discussions start on March 2

slides should be reviewed together with Subarna a week before
then a couple of days before with Stratos

30 minute presentation
Presentations/paper discussions start on March 2

slides should be reviewed together with Subarna a week before
then a couple of days before with Stratos

30 minute presentation

Answer Review Questions

Provide questions for class discussion

Slides follow formatting guidelines
No graphics from paper - your own animated and simple ones
Research projects: slack groups, logistics, midway check in goals, etc as of Thursday
**Research projects:** slack groups, logistics, midway check in goals, etc as of Thursday

**Extra OH:** open discussion in ~2 weeks (poll in piazza)
SEARCHING A MASSIVE DESIGN SPACE

SLOW    OPTIMAL?    ADAPTIVE?
Deep Reinforcement Learning

Neural Net

Cost Estimation/Synthesis

Bayesian Optimization

Genetic Algorithms

Input

Output

Learned Shortcuts

Workload

Hardware

SLAs

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

High confidence

Low confidence

Feedback

Data Layout and Index Synthesis

Data Node (Element) Design Space

Node has partitioning?
Node has Bloom filters?
Node has Zone maps?

Design Continuums

Fast analytical model optimisation

Design Search

Memoization

H/W Pruning

Expert Rules

Reinforcement Learning

Neural Net

Cost Estimation/Synthesis

Design Continuums

arg min f(x)

Node by node design process

Systematically evaluate various designs for each node

Otherwise

Update design

Is it a design continuum?

Binary Search

Range Scan

Equality Scan

Data Access Primitives

Micro-benchmarks

Machine Learning

Micro-benchmarks train models on different hardware profiles.

f(x) = ax + b

Design Space

Data Layout and Index Synthesis

Data Node (Element) Library

Linked List

B-Tree

B+Tree

Trie

Skip List

Array

Data Page

Operation Synthesis

Operation Synthesis (Level 1)

Put

Get

Delete

Bulk Load

Level 1 to Level 2 translation

Concurrency & Updates

Hardware Conscious Synthesis (Level 2)

Put

Get

Delete

Bulk Load

Serial Scan

Equality Scan

Range Scan

Sorted Search

Binary Search

Random Probe

Hardware Profiles

Cost Synthesizer

Machine Learning

Micro-benchmarks

Generalized Cost and Algorithm Synthesis

Space Efficiency Optimization

AST

Code Generator

SLAs

H/W Pruning

Expert Rules

Memoization

Learned Shortcuts

Convergence

Low confidence

High confidence

Feedback

Data structure design

Code

Partial design

Solved

Currently solving

Not yet solved

Initial design

Time threshold

Distance to optimal

Layout Primitives

Overall Designs

Data Access Primitives

Cost Synthesizer

Design Continuums

arg min f(x)

Machine Learning

Micro-benchmarks train models on different hardware profiles.

f(x) = ax + b

Node by node design process

Systematically evaluate various designs for each node

Otherwise

Update design

Is it a design continuum?

Binary Search

Range Scan

Equality Scan

Data Access Primitives

Micro-benchmarks

Machine Learning

Micro-benchmarks train models on different hardware profiles.

f(x) = ax + b

Design Space

Data Layout and Index Synthesis

Data Node (Element) Library

Linked List

B-Tree

B+Tree

Trie

Skip List

Array

Data Page

Operation Synthesis

Operation Synthesis (Level 1)

Put

Get

Delete

Bulk Load

Level 1 to Level 2 translation

Concurrency & Updates

Hardware Conscious Synthesis (Level 2)

Put

Get

Delete

Bulk Load

Serial Scan

Equality Scan

Range Scan

Sorted Search

Binary Search

Random Probe

Hardware Profiles

Cost Synthesizer

Machine Learning

Micro-benchmarks

Generalized Cost and Algorithm Synthesis

Space Efficiency Optimization

AST

Code Generator

SLAs

H/W Pruning

Expert Rules

Memoization

Learned Shortcuts

Convergence

Low confidence

High confidence

Feedback

Data structure design

Code

Partial design

Solved

Currently solving

Not yet solved

Initial design

Time threshold

Distance to optimal

Layout Primitives

Overall Designs

Data Access Primitives

Cost Synthesizer

Design Continuums

arg min f(x)

Machine Learning

Micro-benchmarks train models on different hardware profiles.

f(x) = ax + b

Node by node design process

Systematically evaluate various designs for each node

Otherwise

Update design

Is it a design continuum?

Binary Search

Range Scan

Equality Scan

Data Access Primitives

Micro-benchmarks

Machine Learning

Micro-benchmarks train models on different hardware profiles.

f(x) = ax + b

Design Space

Data Layout and Index Synthesis

Data Node (Element) Library

Linked List

B-Tree

B+Tree

Trie

Skip List

Array

Data Page

Operation Synthesis

Operation Synthesis (Level 1)

Put

Get

Delete

Bulk Load

Level 1 to Level 2 translation

Concurrency & Updates

Hardware Conscious Synthesis (Level 2)

Put

Get

Delete

Bulk Load

Serial Scan

Equality Scan

Range Scan

Sorted Search

Binary Search

Random Probe

Hardware Profiles

Cost Synthesizer

Machine Learning

Micro-benchmarks

Generalized Cost and Algorithm Synthesis

Space Efficiency Optimization

AST

Code Generator

SLAs

H/W Pruning

Expert Rules

Memoization

Learned Shortcuts

Convergence

Low confidence

High confidence

Feedback

Data structure design

Code

Partial design

Solved

Currently solving

Not yet solved

Initial design

Time threshold

Distance to optimal

Layout Primitives

Overall Designs

Data Access Primitives

Cost Synthesizer

Design Continuums

arg min f(x)

Machine Learning

Micro-benchmarks train models on different hardware profiles.

f(x) = ax + b

Node by node design process

Systematically evaluate various designs for each node

Otherwise

Update design

Is it a design continuum?

Binary Search

Range Scan

Equality Scan

Data Access Primitives

Micro-benchmarks

Machine Learning

Micro-benchmarks train models on different hardware profiles.

f(x) = ax + b

Design Space

Data Layout and Index Synthesis

Data Node (Element) Library

Linked List

B-Tree

B+Tree

Trie

Skip List

Array

Data Page

Operation Synthesis

Operation Synthesis (Level 1)

Put

Get

Delete

Bulk Load

Level 1 to Level 2 translation

Concurrency & Updates

Hardware Conscious Synthesis (Level 2)

Put

Get

Delete

Bulk Load

Serial Scan

Equality Scan

Range Scan

Sorted Search

Binary Search

Random Probe

Hardware Profiles

Cost Synthesizer

Machine Learning

Micro-benchmarks

Generalized Cost and Algorithm Synthesis

Space Efficiency Optimization

AST

Code Generator

SLAs

H/W Pruning

Expert Rules

Memoization

Learned Shortcuts

Convergence

Low confidence

High confidence

Feedback

Data structure design

Code
**Deep Reinforcement Learning**

- **Neural Net**
- **Cost Estimation/Synthesis**
- **Feedback**
- **Bayesian Optimization**
- **Genetic Algorithms**

**Input**
- Workload
- Hardware
- SLAs

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

**Learned Shortcuts**
- High confidence
- Low confidence

**Output**
- Data structure design
- AST

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

**Cost Synthesizer**
- Machine Learning
- Micro-benchmarks train models on different hardware profiles.

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

**Space Efficiency Optimization**
- Concurrency & Updates

**Design Space**
- <k, v>

**Data Layout and Index Synthesis**
- Key Value Data Structures
- Data Node (Element) Library
- Design Space

**Data Node (Element) Design Space**
- Node has partitioning?
- Node has Bloom filters?
- Node has Zone maps?

**Layout Primitives**
- Data Node (Element)
- Library
- Design Continuums

**Overall Designs**
- Design Continuums
- Fast analytical model optimisation
- \( \arg \min f(x) \)

**Overall Designs**
- Node by node design process

**Design Search**
- Iterative Search
- Reinforcement Learning
- Deep Reinforcement Learning
- Bayesian Optimization
- Genetic Algorithms

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

**Convergence & Updates**
- Code Generator

**Learned Shortcuts**
- Memoization
- H/W Pruning
- Expert Rules

**Performance**
- H/W Pruning
- Expert Rules
- Memoization

**Code Generator**
- C++

**Data Access Primitives**
- Equality Scan
- Range Scan
- Binary Search

**Design Continuums**
- Is it a design continuum?
- Systematically evaluate various designs for each node

**Classification**
- Learned Shortcuts
- Feedback

**Update design**
- Partial design
- Solved
- Not yet solved

**Convergence & Updates**
- Convergence & Updates

**Input**
- Workload
- Hardware
- SLAs

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

**Learned Shortcuts**
- High confidence
- Low confidence

---

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

**Cost Synthesizer**
- Machine Learning
- Micro-benchmarks train models on different hardware profiles.

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

**Space Efficiency Optimization**
- Concurrency & Updates

**Learned Shortcuts**
- Memoization
- H/W Pruning
- Expert Rules

**Performance**
- H/W Pruning
- Expert Rules
- Memoization

**Code Generator**
- C++

**Data Access Primitives**
- Equality Scan
- Range Scan
- Binary Search

**Cost Synthesizer**
- Machine Learning
- Micro-benchmarks train models on different hardware profiles.

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

**Space Efficiency Optimization**
- Concurrency & Updates
discrete

LSM-tree

great writes
decent reads
space amp

B-tree

good reads
ok updates
space amp
performance continuum?
performance continuum?
performance continuum?
not LSM-tree, not B-tree
mixed design principles
hybrid performance properties
design continuum

not LSM-tree, not B-tree
mixed design principles
hybrid performance properties
totally different?

Level 1

Level 2

Level 3

... ...

Level N

index (fence+pointer)

data
totally different?

Level 1

index+ data

Level 2

index+data

Level 3

index+data

Level N

index (fence+pointer)

data

totally different?
>1 B-trees that have not (yet) been merged
>1 B-trees that have not (yet) been merged

B-tree entries have not been propagated to the leaves
navigation transitions

speed?
LSM-Trees
- size ratio
- merge policy
- filters bits per entry
- size of buffer/cache
- internal k-v layout

B-Trees
- fanout
- node layout
- partitioning policy
- fill factor
- split policy
a unified design space

[log, log+hash, LSM-tree*, BεTree, B-Tree, Sorted Array]
a unified design space

[log, log+hash, LSM-tree*, BεTree, B-Tree, Sorted Array]

design principles for >1 structures
a unified design space
[log, log+hash, LSM-tree*, B^εTree, B-Tree, Sorted Array]

Buffer
Filters
Fence Pointers
Storage
L-Y hot levels
Y cold levels

Memory

Filter Choice by Mem. Budget

Bloom Filter
Hash Table

(or)
(Fewer Bits per Key)
(Full Key Size)
a unified design space

[log, log+hash, LSM-tree*, B^εTree, B-Tree, Sorted Array]

Filter Choice by Mem. Budget

- Bloom Filter: (Fewer Bits per Key)
- Hash Table: (Full Key Size)

Filter Choice by Mem. Budget

Storage

L-Y hot levels

Y cold levels

Memory

Buffer

Filters

Fence Pointers

storage block node

run boundary

fence pointer

oldest hot run contains cascading fence pointers

[log, log+hash, LSM-tree*, B^εTree, B-Tree, Sorted Array]
a unified design space

[log, log+hash, LSM-tree*, B⁺Tree, B-Tree, Sorted Array]

Buffer

Filters

Fence Pointers

storage block node

run boundary fence pointer

Storage

Memory

Filter Choice by Mem. Budget

Bloom Filter

or

Hash Table

(Fewer Bits per Key)

(Full Key Size)

L-Y hot levels

Y cold levels

oldest hot run contains cascading fence pointers
a unified design space

[log, log+hash, LSM-tree*, BεTree, B-Tree, Sorted Array]
a unified design space
[log, log+hash, LSM-tree*, BEC Tree, B-Tree, Sorted Array]

Buffer
Filter Choice by Mem. Budget
(Bloom Filter)
(Fewer Bits per Key)
Hash Table
(Full Key Size)

Filter Choice by Mem. Budget
(Fewer Bits per Key)
(Full Key Size)

Storage

L-Y hot levels

Y cold levels

Fence Pointers
run boundary fence pointer

storage block node

Memory

FP
FP
FP
FP
FP
a unified design space

[log, log+hash, LSM-tree*, BεTree, B-Tree, Sorted Array]
*a unified design space*

[log, log+hash, LSM-tree*, BεTree, B-Tree, Sorted Array]

---

Diagram showing a unified design space with components such as Buffer, Filters, Fence Pointers, and Memory. The diagram illustrates the relationship between Bloom Filter or Hash Table and Filter Choice by Mem. Budget. The storage block node, run boundary, and fence pointer are also depicted.
Lazy Leveled LSM-Tree

Buffer
Filters
Fence Pointers

HT

Memroy
Storage

LSH-Table
### Components

<table>
<thead>
<tr>
<th><strong>Term</strong></th>
<th><strong>Description</strong></th>
<th><strong>Units</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>Block Size</td>
<td>Entries</td>
</tr>
<tr>
<td>M</td>
<td>Memory</td>
<td>Bits</td>
</tr>
<tr>
<td>N</td>
<td>Dataset Size</td>
<td>Entries</td>
</tr>
<tr>
<td>E</td>
<td>Entry Size</td>
<td>Bits</td>
</tr>
<tr>
<td>F</td>
<td>Key Size</td>
<td>Bits</td>
</tr>
<tr>
<td>s</td>
<td>Avg. Selectivity</td>
<td>Entries</td>
</tr>
<tr>
<td>T</td>
<td>Growth Factor</td>
<td>Bits</td>
</tr>
<tr>
<td>K</td>
<td>Hot Merge</td>
<td>Runs</td>
</tr>
<tr>
<td>Z</td>
<td>Cold Merge</td>
<td>Runs</td>
</tr>
<tr>
<td>D</td>
<td>Max. Node Size</td>
<td>Blocks</td>
</tr>
</tbody>
</table>

#### Environment Parameters

<table>
<thead>
<tr>
<th><strong>Derived Term</strong></th>
<th><strong>Expression</strong></th>
<th><strong>Units</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>L (total levels)</td>
<td>$\log_T \left( \frac{N \cdot X^p}{M} \right)$</td>
<td>Levels</td>
</tr>
<tr>
<td>X (Filters)</td>
<td>$\frac{1}{T^p (\frac{\ln T + \ln K}{\ln T} + \frac{\ln Z}{\ln T})}$</td>
<td>Bits per Entry</td>
</tr>
<tr>
<td>MRLO (Hot Levels Saturation)</td>
<td>$N \cdot \left( \frac{1}{Z} + \frac{1}{T} \right)$</td>
<td>Bits</td>
</tr>
<tr>
<td>MF (Hot Levels Saturation)</td>
<td>$\frac{M \cdot F}{T \cdot B}$</td>
<td>Bits</td>
</tr>
<tr>
<td>Y (Cold Levels)</td>
<td>$\left{ \begin{array}{ll} \log_T \left( \frac{N \cdot X^p}{M} \right) &amp; \text{if } MR \geq MRLO \ \frac{1}{L - 1} &amp; \text{if } MRLO &lt; MR &lt; MRH \ \text{if } MR = MRLO \end{array} \right.$</td>
<td>Levels</td>
</tr>
<tr>
<td>MF (Filter Memory Budget)</td>
<td>$M_F - MF$</td>
<td>Bits</td>
</tr>
<tr>
<td>MB (Buffer Memory Budget)</td>
<td>$B \cdot E + (M - M_F)$</td>
<td>Bits</td>
</tr>
<tr>
<td>BF (BF False Positive Rate)</td>
<td>$e^{-\frac{MBF}{N} \cdot \ln^p \cdot T^Y \cdot Z \cdot \frac{1}{Z} \cdot K} \cdot \frac{1}{\ln T}$</td>
<td>Probability</td>
</tr>
</tbody>
</table>

#### Design Parameters

<table>
<thead>
<tr>
<th><strong>Term</strong></th>
<th><strong>Expression</strong></th>
<th><strong>Units</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>BF</td>
<td>Storage block node run boundary fence pointer</td>
<td></td>
</tr>
<tr>
<td>Buffer</td>
<td>Fence Pointers</td>
<td></td>
</tr>
<tr>
<td>Bloom</td>
<td>Filters</td>
<td></td>
</tr>
</tbody>
</table>

#### General Cost Model

<table>
<thead>
<tr>
<th><strong>Operation</strong></th>
<th><strong>Cost Expression (I/O)</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Update</td>
<td>$O(\frac{1}{B} \cdot (\frac{K}{B} \cdot (L - Y - 1) + \frac{T}{Z} \cdot (Y + 1)))$</td>
</tr>
<tr>
<td>Zero Result Lookup</td>
<td>$O(Z \cdot e^{-\frac{MBF}{N} \cdot T^Y} + Y \cdot Z)$</td>
</tr>
<tr>
<td>Single Result Lookup</td>
<td>$O(1 + Z \cdot e^{-\frac{MBF}{N} \cdot T^Y} + Y \cdot Z)$</td>
</tr>
<tr>
<td>Short Scan</td>
<td>$O(K \cdot (L - Y - 1) + Z \cdot (Y + 1))$</td>
</tr>
<tr>
<td>Long Scan</td>
<td>$O(\frac{S \cdot Z}{B})$</td>
</tr>
</tbody>
</table>
**Environment Parameters**

<table>
<thead>
<tr>
<th>Term</th>
<th>Name</th>
<th>Description</th>
<th>Min. Value</th>
<th>Max. Value</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>Block Size</td>
<td># data entries that fit in a storage block.</td>
<td></td>
<td></td>
<td>Entries</td>
</tr>
<tr>
<td>M</td>
<td>Memory</td>
<td>Total main memory budget.</td>
<td>(B \cdot E + \frac{M}{E} \cdot N)</td>
<td>(N \cdot E)</td>
<td>Bits</td>
</tr>
<tr>
<td>N</td>
<td>Data Size</td>
<td>Size of contiguous data region</td>
<td>( \frac{\ln 2}{K} )</td>
<td>( \frac{\ln K}{L} )</td>
<td>Bits</td>
</tr>
<tr>
<td>E</td>
<td>Key Size</td>
<td>Size of a key, also used to approximate size of a fence (fence key and pointer).</td>
<td></td>
<td></td>
<td>Bits</td>
</tr>
<tr>
<td>s</td>
<td>Avg. Selectivity</td>
<td>Average selectivity of a long range query.</td>
<td></td>
<td></td>
<td>Entries</td>
</tr>
<tr>
<td>T</td>
<td>Growth Factor</td>
<td>Capacity ratio between adjacent levels.</td>
<td>2</td>
<td>(B)</td>
<td>Ratio</td>
</tr>
<tr>
<td>K</td>
<td>Hot Merge Threshold</td>
<td>Maximum # runs per hot level.</td>
<td>1</td>
<td>(T - 1)</td>
<td>Runs</td>
</tr>
<tr>
<td>Z</td>
<td>Cold Merge Threshold</td>
<td>Maximum # runs per cold level.</td>
<td>1</td>
<td>(T - 1)</td>
<td>Runs</td>
</tr>
<tr>
<td>D</td>
<td>Max. Node Size</td>
<td>Maximum size of a node, including a contiguous data region.</td>
<td>1</td>
<td>( \frac{N}{B} )</td>
<td>Blocks</td>
</tr>
<tr>
<td>(M_F)</td>
<td>Fence &amp; Filter Memory Budget</td>
<td># bits of main memory budgeted to fence pointers and filters.</td>
<td>(\frac{F \cdot T \cdot M}{E \cdot N})</td>
<td>(M)</td>
<td>Bits</td>
</tr>
</tbody>
</table>

**Design Parameters**

- \(B\): Block Size
- \(M\): Memory
- \(N\): Data Size
- \(E\): Key Size
- \(s\): Avg. Selectivity
- \(T\): Growth Factor
- \(K\): Hot Merge Threshold
- \(Z\): Cold Merge Threshold
- \(D\): Max. Node Size
- \(M_F\): Fence & Filter Memory Budget

**Derived Design Rules**

- **Operation**
  - \(O(B)\)
  - \(O(1 + Z)\)
  - \(O(K)\)
  - \(O(1 + Z)\)

- **Update**
  - \(O(B)\)

- **Zero Result Lookup**
  - \(O(1 + Z)\)

- **Single Result Lookup**
  - \(O(1 + Z)\)

- **Short Scan**
  - \(O(K)\)

- **Long Scan**
  - \(O(K)\)

**General Cost Model**

- \(\text{General Cost Model} = \sum_{i=1}^{N} (\text{Operation}_i + \text{Update}_i + \text{Zero Result Lookup}_i + \text{Single Result Lookup}_i + \text{Short Scan}_i + \text{Long Scan}_i)\)
### Environment Parameters

<table>
<thead>
<tr>
<th>Term</th>
<th>Name</th>
<th>Description</th>
<th>Min. Value</th>
<th>Max. Value</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>Block Size</td>
<td># data entries that fit in a storage block.</td>
<td></td>
<td></td>
<td>Entries</td>
</tr>
<tr>
<td>M</td>
<td>Memory</td>
<td>Total main memory budget.</td>
<td>$B \cdot E + \frac{T \cdot M}{E}$</td>
<td>$N \cdot E$</td>
<td>Bits</td>
</tr>
<tr>
<td>N</td>
<td>Data Size</td>
<td>Maximum number of data entries in the dataset.</td>
<td></td>
<td></td>
<td>Entries</td>
</tr>
<tr>
<td>E</td>
<td>Average Size</td>
<td>Size of a key, also used to approximate size of a fence (fence key and pointer).</td>
<td></td>
<td></td>
<td>Bits</td>
</tr>
<tr>
<td>s</td>
<td>Avg. Selectivity</td>
<td>Average selectivity of a long range query.</td>
<td></td>
<td></td>
<td>Entries</td>
</tr>
</tbody>
</table>

### Design Parameters

- **Growth Factor**: $G = \frac{S_{i+1}}{S_i}$
- **Hot Merge Threshold**: Maximum # runs per hot level.
- **Cold Merge Threshold**: Maximum # runs per cold level.
- **Max Node Size**: Maximum size of a node, i.e. a contiguous data region.
- **Fence & Filter Memory Budget**: # bits of main memory budgeted to fence pointers and filters.

### General Cost Model

- **Derived Term**: $L$ (total levels)
  - $L = \log_2 \left( \frac{N+1}{M} \right)$
- **X (Filters Memory Threshold)**
  - $T = \frac{\ln N}{M}$
  - Filters Memory Threshold
  - $M_{FP}$ (Filter Memory Budget)
- **Y (Cold Levels)**
  - Maximum # cold levels
  - $Y = \frac{\ln \left( \frac{B}{L-1} \right)}{T}$
  - Cold Levels Saturation
  - $M_{CF}$ (Cold Levels Saturation)
- **Z (Hot Levels)**
  - Maximum # hot levels
  - $Z = \frac{\ln \left( \frac{B}{L-1} \right)}{T}$
  - Hot Levels Saturation
  - $M_{HF}$ (Hot Levels Saturation)

### Derived Design Rules

- **Filter Memory Budget**: $M = M_{FP}$
- **Buffer Memory Budget**: $B = \frac{E \cdot M}{1 - \frac{E}{Z}}$
- **BF False Positive Rate**: $P_{BF} = \frac{1}{N} \cdot \frac{E}{Z}$

### Database Models

- **Unified I/O Models**: $O(1 + Z \cdot Y)$
  - Short Scan
  - $O(1 + Z \cdot Y)$
- **Long Scan**: $O(Y)$
- **Update**: $O(\frac{1}{N} \cdot L \cdot Y \cdot \frac{M}{Z} \cdot (Y+1))$
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>T (Growth Factor)</td>
<td>$\frac{N\cdot E}{M_B}$</td>
<td>$\frac{N\cdot E}{M_B}$</td>
<td>[2, $B$]</td>
<td>[2, $B$]</td>
<td>[2, $B$]</td>
<td>[2, $B$]</td>
<td>[2, $B$]</td>
<td>1</td>
<td>1</td>
<td>$B$</td>
</tr>
<tr>
<td>K (Hot Merge Threshold)</td>
<td>$T - 1$</td>
<td>$T - 1$</td>
<td>$T - 1$</td>
<td>$T - 1$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>$B$</td>
</tr>
<tr>
<td>Z (Cold Merge Threshold)</td>
<td>$T - 1$</td>
<td>$T - 1$</td>
<td>$T - 1$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>$B$</td>
</tr>
<tr>
<td>D (Max. Node Size)</td>
<td>1</td>
<td>1</td>
<td>$\frac{N}{T}$</td>
<td>$\frac{N}{T}$</td>
<td>$\frac{N}{T}$</td>
<td>$\frac{N}{T}$</td>
<td>$\frac{N}{T}$</td>
<td>1</td>
<td>1</td>
<td>$\frac{N}{T}$</td>
</tr>
<tr>
<td>$M_F$ (Fence &amp; Filter Mem.)</td>
<td>$\frac{N\cdot E}{M_B}$</td>
<td>$N\cdot F\cdot(1 + \frac{1}{T})$</td>
<td>$N\cdot(\frac{F}{T} + 10)$</td>
<td>$N\cdot(\frac{F}{T} + 10)$</td>
<td>$F\cdot T\cdot M_B \cdot E^{-B}$</td>
<td>$F\cdot T\cdot M_B \cdot E^{-B}$</td>
<td>$F\cdot T\cdot M_B \cdot E^{-B}$</td>
<td>$F\cdot T\cdot M_B \cdot E^{-B}$</td>
<td>$N\cdot E$</td>
<td>$M_B$</td>
</tr>
<tr>
<td>Update</td>
<td>$O(\frac{L}{T})$</td>
<td>$O(\frac{L}{T})$</td>
<td>$O(\frac{L}{T})$</td>
<td>$O(\frac{L}{T})$</td>
<td>$O(\frac{T}{E} \cdot L)$</td>
<td>$O(\frac{T}{E} \cdot L)$</td>
<td>$O(\frac{T}{E} \cdot L)$</td>
<td>$O(\frac{T}{E} \cdot L)$</td>
<td>$O(\frac{T}{E} \cdot L)$</td>
<td>$O(L)$</td>
</tr>
<tr>
<td>Zero Result Lookup</td>
<td>$O(\frac{N\cdot E}{M_B})$</td>
<td>$O(0)$</td>
<td>$O(T \cdot e^{-\frac{M_B F}{N}})$</td>
<td>$O(e^{-\frac{M_B F}{N}})$</td>
<td>$O(\frac{T}{E} \cdot L)$</td>
<td>$O(\frac{T}{E} \cdot L)$</td>
<td>$O(\frac{T}{E} \cdot L)$</td>
<td>$O(\frac{T}{E} \cdot L)$</td>
<td>$O(\frac{T}{E} \cdot L)$</td>
<td>$O(L)$</td>
</tr>
<tr>
<td>Existing Lookup</td>
<td>$O(\frac{N\cdot E}{M_B})$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(L)$</td>
</tr>
<tr>
<td>Short Scan</td>
<td>$O(\frac{N\cdot E}{M_B})$</td>
<td>$O(\frac{N\cdot E}{M_B})$</td>
<td>$O(L \cdot T)$</td>
<td>$O(1 + T \cdot (L - 1))$</td>
<td>$O(L)$</td>
<td>$O(L)$</td>
<td>$O(L)$</td>
<td>$O(L)$</td>
<td>$O(L)$</td>
<td>$O(L)$</td>
</tr>
<tr>
<td>Long Scan</td>
<td>$O(\frac{N\cdot E}{M_B})$</td>
<td>$O(\frac{N\cdot E}{M_B})$</td>
<td>$O(T \cdot \frac{F}{T})$</td>
<td>$O(\frac{F}{T})$</td>
<td>$O(\frac{F}{T})$</td>
<td>$O(\frac{F}{T})$</td>
<td>$O(\frac{F}{T})$</td>
<td>$O(\frac{F}{T})$</td>
<td>$O(\frac{F}{T})$</td>
<td>$O(\frac{F}{T})$</td>
</tr>
</tbody>
</table>
from write to read optimized

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>T (Growth Factor)</td>
<td>( \frac{N \cdot E}{M_B} )</td>
<td>[2, ( B )]</td>
<td>[2, ( B )]</td>
<td>2</td>
<td>[2, ( B )]</td>
<td>( B )</td>
<td>( \frac{N \cdot E}{M_B} )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>K (Hot Merge Threshold)</td>
<td>( T - 1 )</td>
<td>( T - 1 )</td>
<td>( T - 1 )</td>
<td>( T - 1 )</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Z (Cold Merge Threshold)</td>
<td>( T - 1 )</td>
<td>( T - 1 )</td>
<td>( T - 1 )</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>D (Max. Node Size)</td>
<td>1</td>
<td>( [1, \frac{N}{T}] )</td>
<td>( [1, \frac{N}{T}] )</td>
<td>( [1, \frac{N}{T}] )</td>
<td>( \frac{N}{T} )</td>
<td>( \frac{N}{T} )</td>
<td>1</td>
<td>1</td>
<td>( \frac{N}{T} )</td>
<td></td>
</tr>
<tr>
<td>( M_F ) (Fence &amp; Filter Mem.)</td>
<td>( \frac{N \cdot E}{M_B} )</td>
<td>( N \cdot F \cdot (1 + \frac{1}{T}) )</td>
<td>( N \cdot (\frac{1}{T} + 10) )</td>
<td>( N \cdot (\frac{1}{T} + 10) )</td>
<td>( \frac{F \cdot T \cdot M_B}{E \cdot B} )</td>
<td>( \frac{F \cdot T \cdot M_B}{E \cdot B} )</td>
<td>( \frac{F \cdot T \cdot M_B}{E \cdot B} )</td>
<td>( \frac{F \cdot T \cdot M_B}{E \cdot B} )</td>
<td>( \frac{N \cdot E}{M_B} )</td>
<td></td>
</tr>
<tr>
<td>Update</td>
<td>( O(\frac{L}{T}) )</td>
<td>( O(\frac{L}{T}) )</td>
<td>( O(\frac{L}{T}) \cdot (T + L) )</td>
<td>( O(\frac{T}{B} \cdot L) )</td>
<td>( O(\frac{L}{T}) )</td>
<td>( O(\frac{L}{T} \cdot L) )</td>
<td>( O(\frac{L}{T} \cdot L) )</td>
<td>( O(L) )</td>
<td>( O(\frac{N \cdot E}{M_B \cdot B}) )</td>
<td></td>
</tr>
<tr>
<td>Zero Result Lookup</td>
<td>( O(\frac{N \cdot E}{M_B}) )</td>
<td>( O(0) )</td>
<td>( O(T \cdot e^{-\frac{M \cdot F}{N}}) )</td>
<td>( O(e^{-\frac{M \cdot F}{N}}) )</td>
<td>( O(L) )</td>
<td>( O(L) )</td>
<td>( O(L) )</td>
<td>( O(L) )</td>
<td>( O(L) )</td>
<td></td>
</tr>
<tr>
<td>Existing Lookup</td>
<td>( O(\frac{N \cdot E}{M_B}) )</td>
<td>( O(1) )</td>
<td>( O(1 + T \cdot e^{-\frac{M \cdot F}{N}}) )</td>
<td>( O(1) )</td>
<td>( O(L) )</td>
<td>( O(L) )</td>
<td>( O(L) )</td>
<td>( O(L) )</td>
<td>( O(L) )</td>
<td></td>
</tr>
<tr>
<td>Short Scan</td>
<td>( O(\frac{N \cdot E}{M_B} \cdot \frac{1}{T}) )</td>
<td>( O(\frac{N \cdot E}{M_B} \cdot \frac{1}{T}) )</td>
<td>( O(L \cdot T) )</td>
<td>( O(1 + T \cdot (L - 1)) )</td>
<td>( O(L) )</td>
<td>( O(L) )</td>
<td>( O(L) )</td>
<td>( O(L) )</td>
<td>( O(L) )</td>
<td></td>
</tr>
<tr>
<td>Long Scan</td>
<td>( O(\frac{N \cdot E}{M_B} \cdot \frac{1}{T}) )</td>
<td>( O(\frac{N \cdot E}{M_B} \cdot \frac{1}{T}) )</td>
<td>( O(T \cdot \frac{1}{T}) )</td>
<td>( O(\frac{1}{T}) )</td>
<td>( O(\frac{1}{T}) )</td>
<td>( O(\frac{1}{T}) )</td>
<td>( O(\frac{1}{T}) )</td>
<td>( O(\frac{1}{T}) )</td>
<td>( O(\frac{1}{T}) )</td>
<td></td>
</tr>
</tbody>
</table>
Design Primitives

- fanout
- filter bits
- buffer size
- merge policy

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

Data structures

Design continuum

Design space

LSM

B-tree

memory

read

write

DASlab © Harvard SEAS
Deep Reinforcement Learning

Neural Net

Cost Estimation/Synthesis

Bayesian Optimization

Genetic Algorithms

Input

Output

Data Layout and Index Synthesis

Generalized Cost and Algorithm Synthesis

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

Concurrency & Updates

Space Efficiency Optimization

Data Access Primitives

Cost Synthesizer

Machine Learning

Micro-benchmarks train models on different hardware profiles.

\[ f(x) = a \cdot x + b \]

Equality Scan

Range Scan

Binary Search

Hardware Profiles

Operation Synthesis (Level 1)

Put

Get

Delete

Bulk Load

Serial Scan

Equality Scan

Range Scan

Sorted Search

Random Probe

Layout Primitives

Data Node (Element) Library

Key Value Data Structures

< k, v >

Data Node (Element) Design Space

Bloom Filter

Zone Maps

Node has partitioning?

Node has Bloom filters?

Node has Zone maps?

Partitioning

Linked List

B-Tree

Data Page

Data Access Primitives

Design Search

Memoization

H/W Pruning

Expert Rules

Reinforcement Learning

Neural Net

Cost Estimation/Synthesis

Design Search

Input

Output

Learned Shortcuts

Workload

Hardware

SLAs

Performance Constraints

- Initial design
- Time threshold
- Distance to optimal

Learned Shortcuts

High confidence

Low confidence

Feedback

Feedback

Performance Constraints

- Initial design
- Time threshold
- Distance to optimal

Feedback

Feedback

Learned Shortcuts

High confidence

Low confidence

Feedback

Feedback

Learned Shortcuts

High confidence

Low confidence

Feedback

Feedback

Learned Shortcuts

High confidence

Low confidence

Feedback

Feedback

Learned Shortcuts

High confidence

Low confidence

Feedback

Feedback

Learned Shortcuts

High confidence

Low confidence

Feedback

Feedback

Learned Shortcuts

High confidence

Low confidence

Feedback

Feedback

Learned Shortcuts

High confidence

Low confidence

Feedback

Feedback

Learned Shortcuts

High confidence

Low confidence

Feedback

Feedback

Learned Shortcuts

High confidence

Low confidence

Feedback

Feedback

Learned Shortcuts

High confidence

Low confidence

Feedback

Feedback

Learned Shortcuts

High confidence

Low confidence

Feedback

Feedback

Learned Shortcuts

High confidence

Low confidence

Feedback

Feedback

Learned Shortcuts

High confidence

Low confidence

Feedback

Feedback

Learned Shortcuts

High confidence

Low confidence

Feedback

Feedback

Learned Shortcuts

High confidence

Low confidence

Feedback

Feedback
assisted discovery
assisted discovery
assisted discovery

single result lookup cost (I/O)

update cost (I/O)

B^ε-Tree with no order in levels
Tiered LSM with fractional cascading

Tiered LSM with fractional cascading
update cost
(I/O)

single result lookup cost (I/O)

BεTree with no order in levels
Tiered LSM with fractional cascading

BεTree* hot levels
LSM-Tree* cold levels
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
PointLookup \( (searchKey) \)

if \( M_B > E \) then

entry := buffer.find\( (searchKey) \);
if entry then

\[
\text{return entry;}
\]

// Pointer for direct block access. Set to root.
blockToCheck := levels[0].runs[0].nodes[0];
for \( i \leftarrow 0 \) to \( L \) do

// Check each level’s runs from recent to oldest.
for \( j \leftarrow 0 \) to \( \text{levels[i].runs.count} \) do

/* Prune search using bloom filters and fences
when available. */
if \( i < (L - Y) \) // At hot levels.
then

keyCouldExist :=

filters[i][j].checkExists\( (searchKey) \);
if !keyCouldExist then

continue;
else

blockToCheck :=
fences[i][j].find\( (searchKey) \);

/* For oldest hot run, and all cold runs, if no
entry is returned, then the search continues
using a pointer into the next oldest run. */
entry, blockToCheck :=
blockToCheck.find\( (searchKey) \);
if entry then

\[
\text{return entry;}
\]

\[
\text{return keyDoesNotExist;}
\]
Design Continuums and the Path Toward Self-Designing Key-Value Stores that Know and Learn.
cost synthesis across the memory hierarchy (emerging hardware)

graphs, spatial data, images

cloud costs
<table>
<thead>
<tr>
<th>Primitive (Modifier)</th>
<th>Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Concurrency Control</strong></td>
<td><strong>Optimistic</strong>: Operations are made without taking locks with validity checked after the operation and repeated if necessary. <strong>Pessimistic</strong>: Operations are blocked during modification but always succeed when executed. <strong>Lockfree</strong>: operations are made without locks using atomic operations.</td>
</tr>
<tr>
<td><strong>Lock Type</strong></td>
<td>Spin</td>
</tr>
<tr>
<td><strong>Granularity</strong></td>
<td>Global</td>
</tr>
<tr>
<td><strong>Data Rel.</strong></td>
<td><strong>Value Compression</strong></td>
</tr>
<tr>
<td><strong>Examples</strong></td>
<td><strong>Mastree</strong></td>
</tr>
<tr>
<td>Data Ops</td>
<td>SMO</td>
</tr>
<tr>
<td>R</td>
<td>U</td>
</tr>
<tr>
<td>Lockfree</td>
<td>Lockfree</td>
</tr>
<tr>
<td>Spin</td>
<td>Yield[(Try)]</td>
</tr>
<tr>
<td>Global</td>
<td>Hand-over-hand Node</td>
</tr>
<tr>
<td>Invalidate</td>
<td>Removal</td>
</tr>
<tr>
<td>In-place</td>
<td>Out-of-Place</td>
</tr>
<tr>
<td>Delta</td>
<td>Full</td>
</tr>
<tr>
<td>Eagerly</td>
<td>Lazily</td>
</tr>
<tr>
<td>Embedded</td>
<td>Remote</td>
</tr>
<tr>
<td>Fixed width</td>
<td>Variable</td>
</tr>
<tr>
<td>True[(Trie)]</td>
<td>False</td>
</tr>
<tr>
<td>Embedded</td>
<td>Remote</td>
</tr>
<tr>
<td>Bytes</td>
<td></td>
</tr>
<tr>
<td>True(TYPE)</td>
<td>False</td>
</tr>
<tr>
<td>True(TYPE)</td>
<td>False</td>
</tr>
<tr>
<td>True(TYPE)</td>
<td>False</td>
</tr>
<tr>
<td>True(TYPE)</td>
<td>False</td>
</tr>
</tbody>
</table>