Design Tradeoffs of Data Access Methods

Manos Athanassoulis     Stratos Idreos

Harvard University
Part B: Design Dimensions
Partitioning

Logarithmic Design

Fractional Cascading

Log-Structured Updates

Buffering

Differential Updates

Sparse Indexing

RUM

each design decision affects read/update/memory overheads
Partitioning

Logarithmic Design

Fractional Cascading

Log-Structured Updates

Buffering

Differential Updates

Sparse Indexing
Partitioning

Definition

adds structure to the data

helps reads

updates are more expensive (maintain the structure)
Partitioning

Feature Implementation

Range

5

12

22

Hashing

Hash

Radix

2

1

0

0

1

2

1
Partitioning

Feature Implementation

Bounded Disorder

Range

Hash

Hash

Time

epoch 1

1

4

3

epoch 2

19

22

15

epoch 3

9

13

6
Partitioning

... by example

Static

- B-Trees
- Zonemaps
- Hash Index
- Tries
- Bitmap Indexing

Dynamic

- Dynamic Hash Index
- Cracking
- Adaptive Zonemaps
- Adaptive Indexing
Partitioning: a detailed example

a tight column: 

- reads have to scan
- no memory overhead
- in-place updates and efficient inserts
Partitioning: a detailed example

a tight column:

- **reads** have to scan
- no **memory** overhead
- in-place **updates** and efficient inserts

8 2 1 7 6 9 3

a tight sorted column:

- very efficient **reads** (logarithmic search)
- no **memory** overhead
- **updates** & inserts reorganization

1 2 3 6 7 8 9
Partitioning: a detailed example

a tight column: [8 2 1 7 6 9 3]
- reads have to scan
- no memory overhead
- in-place updates and efficient inserts

adding clustering: [2 1 3 7 6 9 8]
- efficient reads
- small memory overhead
- updates & inserts: reorganization

a tight sorted column: [1 2 3 6 7 8 9]
- very efficient reads (logarithmic search)
- no memory overhead
- updates & inserts reorganization
Partitioning: a detailed example

a tight column: 8 2 1 7 6 9 3
- **reads** have to scan
- no **memory** overhead
- in-place **updates** and efficient inserts

a tight sorted column: 1 2 3 6 7 8 9
- very efficient **reads** (logarithmic search)
- no **memory** overhead
- **updates** & inserts reorganization

adding clustering: 2 1 3 7 6 9 8
- efficient **reads**
- small **memory** overhead
- **updates** & inserts: reorganization

... and ghost values: 2 1 3 7 6 9 8
- efficient **reads**
- small **memory** overhead (but increased)
- **updates**: reorganization (but inserts for free)
## Design Opportunity

<table>
<thead>
<tr>
<th>Base Data &amp; Columns</th>
<th>Partitioning</th>
<th>Logarithmic Design</th>
<th>Fractional Cascading</th>
<th>Log-Structured</th>
<th>Buffering</th>
<th>Differential Updates</th>
<th>Sparse Indexing</th>
</tr>
</thead>
<tbody>
<tr>
<td>no, range</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Trees</td>
<td>range, radix, time</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hashing</td>
<td>hash</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bitmaps</td>
<td>range, radix</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Differential Files</td>
<td>time, range</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

What else is needed to "come up" with access methods?
Partitioning

Log-Structured Updates

Sparse Indexing

Logarithmic Design

Fractional Cascading

Buffering

Differential Updates
Logarithmic Design

Definition

organize metadata in an exponentially increasing manner

helps reads (logarithmic search)

helps updates (update in place/amortize update cost)

... at the expense of the metadata
Logarithmic Design

Feature Implementation

connected levels

Traditional Tree Structures

B-Trees & Variants

Tries & Variants

Tree-Trie hybrids
Logarithmic Design

Feature Implementation

independent levels

Update-optimized data organization:

LSM Trees & Variants
MaSM

FD-Tree
Stepped-Merge
## Design Opportunity

<table>
<thead>
<tr>
<th>Partitioning</th>
<th>Logarithmic Design</th>
<th>Fractional Cascading</th>
<th>Log-Structured Buffering</th>
<th>Differential Updates</th>
<th>Sparse Indexing</th>
</tr>
</thead>
<tbody>
<tr>
<td>B-Trees &amp; Variants [1]</td>
<td>range</td>
<td>✓ (naturally)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tries &amp; Variants [2]</td>
<td>radix</td>
<td>✓ (naturally)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LSM-Trees &amp; Variants [3]</td>
<td>time</td>
<td>✓ (naturally)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>


Partitioning

Log-Structured Updates

Sparse Indexing

Logarithmic Design

Buffering

Fractional Cascading

Differential Updates
Fractional Cascading

Definition

adds metadata for efficient accessing/searching
pointers between different "levels" of access methods
easy navigation to the "corresponding" partitions
need maintenance on updates
Fractional Cascading

Feature Implementation

Naturally exists in connected levels!
Fractional Cascading

An additional layer of metadata for independent levels
# Design Opportunity

<table>
<thead>
<tr>
<th>Partitioning</th>
<th>Logarithmic Design</th>
<th>Fractional Cascading</th>
<th>Log-Structured</th>
<th>Buffering</th>
<th>Differential Updates</th>
<th>Sparse Indexing</th>
</tr>
</thead>
<tbody>
<tr>
<td>B-Trees &amp; Variants</td>
<td>range</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tries &amp; Variants</td>
<td>radix</td>
<td>✔</td>
<td>✔ (naturally)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LSM-Trees &amp; Variants</td>
<td>time</td>
<td>✔</td>
<td>✔</td>
<td>[1]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

[1] FD-Tree (PVLDB 2010), bLSM (SIGMOD 2012)
Partitioning

Logarithmic Design

Fractional Cascading

Log-Structured Updates

Buffering

Differential Updates

Sparse Indexing
Log-Structured Updates

apply and organize updates by

**appending** instead of in-place updates

**reads** need to merge updates with old data

\[ R \uparrow U \downarrow M \uparrow \]
Log-Structured Updates

Feature Implementation
## Design Opportunity

<table>
<thead>
<tr>
<th>Partitioning</th>
<th>Logarithmic Design</th>
<th>Fractional Cascading</th>
<th>Log-Structured Buffering</th>
<th>Differential Updates</th>
<th>Sparse Indexing</th>
</tr>
</thead>
<tbody>
<tr>
<td>B-Trees &amp; Variants</td>
<td>range</td>
<td>✓</td>
<td>≈</td>
<td></td>
<td>[1]</td>
</tr>
<tr>
<td>Tries &amp; Variants</td>
<td>radix</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LSM-Trees &amp; Variants</td>
<td>time</td>
<td>✓</td>
<td>≈</td>
<td>✓ (naturally)</td>
<td></td>
</tr>
<tr>
<td>Differential Files</td>
<td>time, range</td>
<td></td>
<td></td>
<td></td>
<td>[2]</td>
</tr>
</tbody>
</table>

Fractional cascading and log-structured updates? **challenging to combine efficiently**

Log-structured updates with radix/hash partitioning? **open research question!**

---

[1] Storage/Memory-Aware Trees: Bw-Tree (ICDE 2013), μ-Tree (EMSOFT 2007), IPLB⁺-Tree (JISE 2011)


✓ integral part of design

≈ optional design decision
Partitioning

Logarithmic Design

Fractional Cascading

Log-Structured Updates

Buffering

Sparse Indexing

Differential Updates
Buffering

Definition

explicitly buffer recently read / updated objects / requests

direct tradeoff between memory and read/update performance

\( R \downarrow U - M \uparrow \) or \( R - U \downarrow M \uparrow \)
Buffering

Recent Reads/Updates/Requests

Feature Implementation

Buffering Recent Updates

Updates
## Design Opportunity

<table>
<thead>
<tr>
<th></th>
<th>Partitioning</th>
<th>Logarithmic Design</th>
<th>Fractional Cascading</th>
<th>Log-Structured</th>
<th>Buffering</th>
<th>Differential Updates</th>
<th>Sparse Indexing</th>
</tr>
</thead>
<tbody>
<tr>
<td>B-Trees &amp; Variants</td>
<td>range</td>
<td>✔</td>
<td>✔</td>
<td>≈</td>
<td></td>
<td>reads [1]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>updates [2]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>requests [3]</td>
<td></td>
</tr>
<tr>
<td>Tries &amp; Variants</td>
<td>radix</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LSM-Trees &amp; Variants</td>
<td>time</td>
<td>✔</td>
<td></td>
<td>≈</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>updates [4]</td>
<td></td>
</tr>
<tr>
<td>Differential Files</td>
<td>time, range</td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>


✔ integral part of design

≈ optional design decision
Partitioning

Logarithmic Design

Fractional Cascading

Log-Structured Updates

Buffering

Differential Updates

Sparse Indexing
Differential Updates

Definition

next step for log-structure

only deltas are stored in order to minimize storage overheads

\[ \delta \]

\[ R \uparrow U \downarrow M \downarrow \]
Differential Updates

Feature Implementation

Query result

Merging

Base Data

Delta Store

log-based

stores data physical location info

tree-based
## Design Opportunity

<table>
<thead>
<tr>
<th>Partitioning</th>
<th>Logarithmic Design</th>
<th>Fractional Cascading</th>
<th>Log-Structured</th>
<th>Buffering</th>
<th>Differential Updates</th>
<th>Sparse Indexing</th>
</tr>
</thead>
<tbody>
<tr>
<td>B-Trees &amp; Variants</td>
<td>range ✔</td>
<td>≈</td>
<td>≈</td>
<td>≈</td>
<td>[1]</td>
<td></td>
</tr>
<tr>
<td>Tries &amp; Variants</td>
<td>radix ✔</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LSM-Trees &amp; Variants</td>
<td>time ✔</td>
<td>≈</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Differential Files</td>
<td>time, range</td>
<td></td>
<td>✔</td>
<td>✔</td>
<td>[2]</td>
<td></td>
</tr>
</tbody>
</table>


✔ integral part of design
≈ optional design decision
Partitioning
Log-Structured Updates
Sparse Indexing
Logarithmic Design
Fractional Cascading
Buffering
Differential Updates
Sparse Indexing

Definition

light-weight indexing that allows for skipping unnecessary data

\( R \downarrow U? M \uparrow \)
Sparse Indexing

membership tests

Data

Feature Implementation

sparse range partitioning

Data

bitwise representation
### Design Opportunity

<table>
<thead>
<tr>
<th>Partitioning</th>
<th>Logarithmic Design</th>
<th>Fractional Cascading</th>
<th>Log-Structured Buffering</th>
<th>Differential Updates</th>
<th>Sparse Indexing</th>
</tr>
</thead>
<tbody>
<tr>
<td>B-Trees &amp; Variants</td>
<td>range</td>
<td>✓</td>
<td>✓</td>
<td>≈</td>
<td>≈</td>
</tr>
<tr>
<td>Tries &amp; Variants</td>
<td>radix</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LSM-Trees &amp; Variants</td>
<td>time</td>
<td>✓</td>
<td>≈</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Differential Files</td>
<td>time, range</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Membership Tests</td>
<td>—</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Zonemaps &amp; Variants</td>
<td>range</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bitmaps &amp; Variants</td>
<td>range</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

[1] BF-Tree (VLDB 2014)


✓ integral part of design
≈ optional design decision
## Design Opportunity

<table>
<thead>
<tr>
<th></th>
<th>Partitioning</th>
<th>Logarithmic</th>
<th>Fractional</th>
<th>Log-Structured</th>
<th>Buffering</th>
<th>Differential Updates</th>
<th>Sparse Indexing</th>
</tr>
</thead>
<tbody>
<tr>
<td>B-Trees &amp; Variants</td>
<td>range</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tries &amp; Variants</td>
<td>radix</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LSM-Trees &amp; Variants</td>
<td>time</td>
<td>✔</td>
<td>≈</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Differential Files</td>
<td>time, range</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>Membership Tests</td>
<td>—</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>Zonemaps &amp; Variants</td>
<td>range</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>Bitmaps &amp; Variants</td>
<td>range</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>Hashing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Base Data &amp; Columns</td>
<td>no, range</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

✔ integral part of design
≈ optional design decision
## Design Opportunity

<table>
<thead>
<tr>
<th>Partitioning</th>
<th>Logarithmic Design</th>
<th>Fractional Cascading</th>
<th>Log-Structured</th>
<th>Buffering</th>
<th>Differential Updates</th>
<th>Sparse Indexing</th>
</tr>
</thead>
<tbody>
<tr>
<td>B-Trees &amp; Variants</td>
<td>range</td>
<td>✔</td>
<td>✔</td>
<td>≈</td>
<td>≈</td>
<td>≈</td>
</tr>
<tr>
<td>Tries &amp; Variants</td>
<td>radix</td>
<td>✔</td>
<td>✔</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>LSM-Trees &amp; Variants</td>
<td>time</td>
<td>✔</td>
<td>≈</td>
<td>✔</td>
<td>✔</td>
<td>?</td>
</tr>
<tr>
<td>Differential Files</td>
<td>time, range</td>
<td>?</td>
<td>?</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>

Open research questions!

✔ integral part of design  
≈ optional design decision
## Design Opportunity

<table>
<thead>
<tr>
<th>Partitioning</th>
<th>Logarithmic Design</th>
<th>Fractional Cascading</th>
<th>Log-Structured</th>
<th>Buffering</th>
<th>Differential Updates</th>
<th>Sparse Indexing</th>
</tr>
</thead>
<tbody>
<tr>
<td>B-Trees &amp; Variants</td>
<td>range</td>
<td></td>
<td>≈</td>
<td>≈</td>
<td>≈</td>
<td>≈</td>
</tr>
<tr>
<td>Tries &amp; Variants</td>
<td>radix</td>
<td></td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>LSM-Trees &amp; Variants</td>
<td>time</td>
<td></td>
<td>≈</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>Differential Files</td>
<td>time, range</td>
<td></td>
<td>✔</td>
<td>✔</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>Membership Tests</td>
<td>—</td>
<td></td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Zonemaps &amp; Variants</td>
<td>range</td>
<td></td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>Bitmaps &amp; Variants</td>
<td>range</td>
<td></td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>Hashing</td>
<td>hash</td>
<td></td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>Base Data &amp; Columns</td>
<td>no, range</td>
<td></td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>

Open research questions!

✔ integral part of design  ≈ optional design decision
Hardware-Aware Access Methods

- Read vs. Write latency
- Impact of Read vs. Write
- Variable latency (due to data placement)

**How?**

Use design elements to *match* hardware properties!

**Examples**

- Partitioning: ensure local (faster) accesses
- Log-Structure/Differential Updates: storage friendly updates
- Buffering: exploit additional memory
Workload-Driven Access Methods

workload-driven orthogonal to design elements

a way to incrementally reach the goal of a design element

can be a design element!
<table>
<thead>
<tr>
<th></th>
<th>Partitioning</th>
<th>Logarithmic Design</th>
<th>Fractional Cascading</th>
<th>Log-Structured</th>
<th>Buffering</th>
<th>Differential Updates</th>
<th>Sparse Indexing</th>
<th>Adaptivity</th>
</tr>
</thead>
<tbody>
<tr>
<td>B-Trees &amp; Variants</td>
<td>range</td>
<td>✓</td>
<td>✓</td>
<td>≈</td>
<td>≈</td>
<td>≈</td>
<td>≈</td>
<td>?</td>
</tr>
<tr>
<td>LSM-Trees &amp; Variants</td>
<td>time</td>
<td>✓</td>
<td>≈</td>
<td>✓</td>
<td>✓</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>Differential Files</td>
<td>time/range</td>
<td>?</td>
<td>?</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>

✓ integral part of design  
≈ optional design decision
<table>
<thead>
<tr>
<th></th>
<th>Partitioning</th>
<th>Logarithmic Design</th>
<th>Fractional Cascading</th>
<th>Log-Structured</th>
<th>Buffering</th>
<th>Differential Updates</th>
<th>Sparse Indexing</th>
<th>Adaptivity</th>
</tr>
</thead>
<tbody>
<tr>
<td>B-Trees &amp; Variants</td>
<td>range</td>
<td>✓</td>
<td>✓</td>
<td>≈</td>
<td>≈</td>
<td>≈</td>
<td>≈</td>
<td>[1]</td>
</tr>
<tr>
<td>LSM-Trees &amp; Variants</td>
<td>time</td>
<td>✓</td>
<td>≈</td>
<td>✓</td>
<td>✓</td>
<td>□</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>Differential Files</td>
<td>time/range</td>
<td>?</td>
<td>?</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>□</td>
<td>?</td>
</tr>
</tbody>
</table>


✓ integral part of design
≈ optional design decision
<table>
<thead>
<tr>
<th></th>
<th>Partitioning</th>
<th>Logarithmic Design</th>
<th>Fractional Cascading</th>
<th>Log-Structured</th>
<th>Buffering</th>
<th>Differential Updates</th>
<th>Sparse Indexing</th>
<th>Adativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>B-Trees &amp; Variants</td>
<td>range</td>
<td>✓</td>
<td>✓</td>
<td>≈</td>
<td>≈</td>
<td>≈</td>
<td>≈</td>
<td>[1]</td>
</tr>
<tr>
<td>LSM-Trees &amp; Variants</td>
<td>time</td>
<td>✓</td>
<td>≈</td>
<td>✓</td>
<td>✓</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>Differential Files</td>
<td>time/range</td>
<td>?</td>
<td>?</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>


✔ integral part of design

≈ optional design decision
<table>
<thead>
<tr>
<th>Partitioning</th>
<th>Logarithmic Design</th>
<th>Fractional Cascading</th>
<th>Log-Structured</th>
<th>Buffering</th>
<th>Differential Updates</th>
<th>Sparse Indexing</th>
<th>Adaptivity</th>
</tr>
</thead>
<tbody>
<tr>
<td>B-Trees &amp; Variants</td>
<td>range</td>
<td>✓</td>
<td>✓</td>
<td>≈</td>
<td>≈</td>
<td>≈</td>
<td>≈</td>
</tr>
<tr>
<td>Tries &amp; Variants</td>
<td>radix</td>
<td>✓</td>
<td>✓</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>LSM-Trees &amp; Variants</td>
<td>time</td>
<td>✓</td>
<td>≈</td>
<td>✓</td>
<td>✓</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>Differential Files</td>
<td>time/range</td>
<td>?</td>
<td>?</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>?</td>
</tr>
<tr>
<td>Bitmaps &amp; Variants</td>
<td>range</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>≈</td>
<td>✓</td>
<td>≈</td>
</tr>
<tr>
<td>Base Data &amp; Columns</td>
<td>no, range</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>≈</td>
</tr>
</tbody>
</table>

map existing designs – find commonalities

propose new combinations and predict their behavior

tune existing access methods (altering/adding individual design elements)

✓ integral part of design
≈ optional design decision

48
DATA SYSTEMS LABORATORY
@ Harvard School of Engineering and Applied Sciences

Designing data systems for the big data era

http://daslab.seas.harvard.edu/

thank you!