Dremel: Interactive Analysis of Web-Scale Datasets
Interactive Queries on Large Data

• Input/Output
  - Sequentially reading a Terabyte from disk in a second requires ~20,000 parallel reads!

• Processing
  - CPU-intensive queries may need to run on thousands of cores to complete within a second.
  - Dealing with failures and stragglers is essential.
Interactive Queries on Large Data

• Input/Output
  - Sequentially reading a Terabyte from disk in a second requires \( \sim 20,000 \) parallel reads! [Nested Columnar Storage]

• Processing
  - CPU-intensive queries may need to run on thousands of cores to complete within a second.
  - Dealing with failures and stragglers is essential.
Interactive Queries on Large Data

• Input/Output
  - Sequentially reading a Terabyte from disk in a second requires ~20,000 parallel reads! [Nested Columnar Storage]

• Processing
  - CPU-intensive queries may need to run on thousands of cores to complete within a second. [Hierarchical Query Processing]
  - Dealing with failures and stragglers is essential.
Interactive Queries on Large Data

• Input/Output
  - Sequentially reading a Terabyte from disk in a second requires ~20,000 parallel reads! [Nested Columnar Storage]

• Processing
  - CPU-intensive queries may need to run on thousands of cores to complete within a second. [Hierarchical Query Processing]
  - Dealing with failures and stragglers is essential. [Profiles, Duplicates or Ignores Them]
## Nested Columnar Storage

<table>
<thead>
<tr>
<th>DocId:</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Links</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Forward:</strong></td>
<td>20</td>
</tr>
<tr>
<td><strong>Name</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Language</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Code:</strong></td>
<td>'en-us'</td>
</tr>
<tr>
<td><strong>Country:</strong></td>
<td>'us'</td>
</tr>
<tr>
<td><strong>Url:</strong></td>
<td>'<a href="http://A">http://A</a>'</td>
</tr>
<tr>
<td><strong>Name</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Url:</strong></td>
<td>'<a href="http://B">http://B</a>'</td>
</tr>
</tbody>
</table>
Nested Columnar Storage

Read Less; Cheaper Decompression!
Nested Columnar Storage

```protobuf
text = "message Document {
  required int64 DocId;
  optional group Links {
    repeated int64 Backward;
    repeated int64 Forward;
  }
  repeated group Name {
    repeated group Language {
      required string Code;
      optional string Country;
    }
    optional string Url;
  }
}
"
```
Nested Columnar Storage

<table>
<thead>
<tr>
<th>DocId</th>
<th>value</th>
<th>r</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name.Url</th>
<th>value</th>
<th>r</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><a href="http://A">http://A</a></td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td><a href="http://B">http://B</a></td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>NULL</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Links.Forward</th>
<th>value</th>
<th>r</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forward</td>
<td>20</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Forward</td>
<td>40</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Forward</td>
<td>60</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name.Language.Code</th>
<th>value</th>
<th>r</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>en-us</td>
<td>0</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>en</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>NULL</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>en-gb</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name.Language.Country</th>
<th>value</th>
<th>r</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>us</td>
<td>0</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>NULL</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>NULL</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>gb</td>
<td>1</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

DocId: 10
Links
Forward: 20
Forward: 40
Forward: 60
Name
Language
Code: 'en-us'
Country: 'us'
Language
Code: 'en'
Url: 'http://A'
Name
Url: 'http://B'
Name
Language
Code: 'en-gb'
Country: 'gb'
Building Columns

<table>
<thead>
<tr>
<th>Name</th>
<th>Language</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>en-us</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Repetition (r) and definition (d) levels encode the structural *delta* between the current value and the previous value.

**(r):** Length of common path prefix

**(d):** Number of fields in the path that could be optional but are actually present
Building Columns

<table>
<thead>
<tr>
<th>Name.Language.Code</th>
<th>value</th>
<th>r</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>en-us</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>en</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

DocId: 10
Links
Forward: 20
Forward: 40
Forward: 60
Name
Language
Code: 'en-us'
Country: 'us'
Language
Code: 'en'
Url: 'http://A'
Name
Url: 'http://B'
Name
Language
Code: 'en-gb'
Country: 'gb'

DocId: 20
Links
Backward: 10
Backward: 30
Forward: 80
Name
Url: 'http://C'
Building Columns

<table>
<thead>
<tr>
<th>Name.Language.Code</th>
<th>value</th>
<th>r</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>en-us</td>
<td>0</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>en</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>NULL</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

- **r₁**: DocId: 10
  - Links:
    - Forward: 20
    - Forward: 40
    - Forward: 60
  - Name:
    - Language:
      - Code: 'en-us'
      - Country: 'us'
    - Url: 'http://A'

- **r₂**: DocId: 20
  - Links:
    - Backward: 10
    - Backward: 30
    - Forward: 80
  - Name:
    - Url: 'http://B'

- **r₂**: DocId: 20
  - Links:
    - Backward: 10
    - Backward: 30
    - Forward: 80
  - Name:
    - Url: 'http://C'

- **r₂**: DocId: 20
  - Links:
    - Backward: 10
    - Backward: 30
    - Forward: 80
  - Name:
    - Url: 'http://C'
Building Columns

<table>
<thead>
<tr>
<th>Name.Language.Code</th>
<th>value</th>
<th>r</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>en-us</td>
<td>0</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>en</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>NULL</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>en-gb</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

DocId: 10
Links
  - Forward: 20
  - Forward: 40
  - Forward: 60

Name
  - Language
    - Code: 'en-us'
    - Country: 'us'
  - Language
    - Code: 'en'
  - Url: 'http://A'

Name
  - Url: 'http://B'

Name
  - Language
    - Code: 'en-gb'
    - Country: 'gb'

DocId: 20
Links
  - Backward: 10
  - Backward: 30
  - Forward: 80

Name
  - Url: 'http://C'
Building Columns

<table>
<thead>
<tr>
<th>Name.Language.Code</th>
<th>value</th>
<th>r</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>en-us</td>
<td>0</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>en</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>NULL</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>en-gb</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>NULL</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

- **r1**
  - **DocId:** 10
  - **Links**
    - Forward: 20
    - Forward: 40
    - Forward: 60
  - **Name**
    - Language Code: 'en-us'
    - Country: 'us'
  - **Url:** 'http://A'

- **r2**
  - **DocId:** 20
  - **Links**
    - Backward: 10
    - Backward: 30
    - Forward: 80
  - **Name**
    - Url: 'http://A'
    - Language Code: 'en-gb'
    - Country: 'gb'
  - **Url:** 'http://B'
Retrieving Columns

Diagram showing the relationships between different columns:

- **DocId**
  - Links.Backward
    - Name.Language.Code
      - Name.Url
      - Name.Language.Country
    - Links.Forward
      - 0,1,2

- Links.Backward
  - 0

- Links.Forward
  - 1
Retrieving Columns

- **DocId**: 0
- **Name.Language.Code**: 0, 1, 2
- **Name.Language.Country**: 0, 1
- **Links.Backward**: 0
- **Links.Forward**: 0
- **Name.Url**: 0
Retrieving Columns

DocId

<table>
<thead>
<tr>
<th>DocId</th>
<th>value</th>
<th>r</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Name.Language.Country

<table>
<thead>
<tr>
<th>value</th>
<th>r</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>us</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>NULL</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>NULL</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>gb</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>
Retrieving Columns

<table>
<thead>
<tr>
<th>DocId</th>
<th>value</th>
<th>r</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name.Language.Country</th>
<th>value</th>
<th>r</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>us</td>
<td>3</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>NULL</td>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>NULL</td>
<td>1</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>gb</td>
<td>3</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Hierarchical Query Processing

client

root server

intermediate servers

leaf servers (with local storage)

storage layer (e.g., GFS)
Hierarchical Query Processing

• Optimized for Select-Project-Aggregate queries.
  - Single Scan over Data
  - Recursive Reducers

• Defers discussion of joins, indexing, updates etc. to future work.

• Scheduler’s Secret Sauce.
Duplicate/Ignore Stragglers

percentage of processed tablets

processing time per tablet (sec)

Duplicates or Ignores Stragglers
Comments/Critiques
Does Dremel really require a new execution engine?

![Bar chart showing execution time (sec) on 3000 nodes. The chart compares 'MR-records' with 87 TB, 'MR-columns' with 0.5 TB, and 'Dremel' with 0.5 TB. The execution times decrease as the amount of data decreases.]
What’s really novel about Aggregation Trees?

- Very similar to the MapReduce model (Leaf servers run Map tasks and Aggregators are Reduce tasks)
- Partial Aggregates/Recursive Reducers have already been proposed by Traditional Databases as well as SCOPE/Dryad.
Can we make other tradeoffs?

- **Input/Output**
  - Sequentially reading a Terabyte from disk in a second requires ~20,000 parallel reads!

- **Processing**
  - CPU-intensive queries may need to run on thousands of cores to complete within a second.
  - Dealing with failures and stragglers is essential.
Can we make other tradeoffs?

• Input/Output
  - Sequentially reading a Terabyte from disk in a second requires ~20,000 parallel reads! [Sampling? In-memory RDDs?]

• Processing
  - CPU-intensive queries may need to run on thousands of cores to complete within a second.
  - Dealing with failures and stragglers is essential.
Can we make other tradeoffs?

• Input/Output
  - Sequentially reading a Terabyte from disk in a second requires ~20,000 parallel reads! [Sampling? In-memory RDDs?]

• Processing
  - CPU-intensive queries may need to run on thousands of cores to complete within a second. [Better Data Partitioning?]
  - Dealing with failures and stragglers is essential.
Can we make other tradeoffs?

- **Input/Output**
  - Sequentially reading a Terabyte from disk in a second requires ~20,000 parallel reads! [Sampling? In-memory RDDs?]

- **Processing**
  - CPU-intensive queries may need to run on thousands of cores to complete within a second. [Better Data Partitioning?]
  - Dealing with failures and stragglers is essential. [Giving Answers with Bounded Errors/Confidence Intervals?]
Thank You!