Dremel: Interactive Analysis of Web-Scale Datasets

Sergey Melnik, Andrey Gubarev, Jing Jing Long, Geoffrey Romer, Shiva Shivakumar, Matt Tolton, Theo Vassilakis

Presenter: MoHan Zhang

*Some images in the presentation are taken from slides made by the original authors.
Outline

• Introduction
• Nested Columnar Storage
• Query Processing
• Experiments and Observations
Outline

• Introduction
• Nested Columnar Storage
• Query Processing
• Experiments and Observations
What is Dremel?

A brand of rotary tools used in the metalworking industry, primarily relying on their speed as opposed to torque...
Dremel is a **Scalable, Interactive ad-hoc query system** for analysis of **large-scale** read-only nested data

- Developed and used by Google since 2006
Key Ideas

• Focuses on achieving interactive speed for very large datasets
  • Multi-Terabyte data, scales to 1000s of nodes

• Uses nested data model with SQL-like language

• Columnar storage format

• Employs tree architecture for query processing
Uses inside Google

• Analysis of crawled web documents.

• Tracking install data for applications on Android Market.

• Crash reporting for Google products.

• OCR results from Google Books.

• Spam analysis.

• Debugging of map tiles on Google Maps.

• Tablet migrations in managed Bigtable instances.

• Results of tests run on Google’s distributed build system.

• Disk I/O statistics for hundreds of thousands of disks.

• Resource monitoring for jobs run in Google’s data centers.

• Symbols and dependencies in Google’s codebase.
Sample Workflow

• Data engineer runs a Map Reduce to find signals from web pages, returning billions of records

• The engineer launches Dremel and runs interactive commands
  
  DEFINE TABLE t AS /path/to/data/*
  SELECT TOP(signal1, 100), COUNT(*) FROM t

• More MR-based processing of the data
Outline

• Introduction
• Nested Columnar Storage
• Query Processing
• Experiments and Observations
Record vs. Columnar Representation

Challenges:
- Lossless representation of nested record structure
- Reconstruct original structure from a subset of fields
Sample Nested Data Model

message Document {
  required int64 DocId; [1,1]
  optional group Links {
    repeated int64 Backward; [0,*]
    repeated int64 Forward;
  }
  repeated group Name {
    repeated group Language {
      required string Code;
      optional string Country; [0,1]
    }
    optional string Url;
  }
}

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'
## Column-Striped Representation

<table>
<thead>
<tr>
<th>DocId</th>
<th>Name.Url</th>
<th>Links.Forward</th>
<th>Links.Backward</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>value</td>
<td>r</td>
<td>d</td>
</tr>
<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>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>en-us</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>en</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>NULL</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>en-gb</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>NULL</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Each column stored as set of blocks
Repetition & Definition Levels

• **Repetition Level:**
  • at what repeated field in the field’s path the value has repeated

• **Definition Levels:**
  • how many fields that could be undefined (optional/repeated) that are actually present in the record
Repetition & Definition Levels

\[ r = 1 \quad r = 2 \quad \text{(non-repeating)} \]

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

**Name.Language.Code**

- **r**: At what repeated field in the field’s path the value has repeated
- **d**: How many fields that could be undefined (opt. or rep.) are actually present

**Example**

- **Language (r=2) has repeated**
- **Name (r=1) has repeated, Name (d=1) is defined**
- **No value: Name (r=0) has repeated, Name is defined (d=1)**

**DocId: 10**

**Links**
- Forward: 20
- Forward: 40
- Forward: 60

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

**DocId: 20**

**Links**
- Backward: 10
- Backward: 30
- Forward: 80

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

**DocId: 20**

**Links**
- **Backward**: 10
- **Forward**: 80

**Name**
- **Url**: 'http://C'
- **Language**
  - Code: 'en-gb'
  - **Country**: 'gb'
Record Assembly

- Goal: Given subset of fields, reconstruct the original records as if they only contained the selected fields.

- Finite State Machine reads the field values and levels for each field and appends the values sequentially to the output records.

Transitions labeled with repetition levels.
Record Assembly from Two Fields

Preserves structure of the parent fields
Outline

• Introduction
• Nested Columnar Storage
• Query Processing
• Experiments and Observations
Sample Query

```sql
SELECT DocId AS Id,
       COUNT(Name.Language.Code) WITHIN Name AS Cnt,
       Name.Url + ',' + Name.Language.Code AS Str
FROM t
WHERE REGEXP(Name.Url, '^http') AND DocId < 20;
```

Output table

<table>
<thead>
<tr>
<th>Id: 10</th>
<th>t₁</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Cnt: 2</td>
</tr>
<tr>
<td></td>
<td>Language</td>
</tr>
<tr>
<td></td>
<td>Str: '<a href="http://A,en-us">http://A,en-us</a>'</td>
</tr>
<tr>
<td></td>
<td>Str: '<a href="http://A,en">http://A,en</a>'</td>
</tr>
<tr>
<td>Name</td>
<td>Cnt: 0</td>
</tr>
</tbody>
</table>

Output schema

```protobuf
message QueryResult {
  required int64 Id;
  repeated group Name {
    optional uint64 Cnt;
    repeated group Language {
      optional string Str;
    }
  }
}
```
Serving Tree Architecture

- Root server: receives incoming queries, reads metadata from tables, and routes queries to the next level
- Intermediate server: parallel aggregation of partial results
- Leaf server: communicate with storage layer / access the data on local disk
Serving Tree

• Designed for aggregate queries returning small-medium results (< 1M), larger aggregations rely on parallel DBMS and Map Reduce

• Query Dispatcher provides scheduling and fault tolerance
  • schedules queries based on their priorities and balances the load
  • If one node becomes much slower, reschedule

• Some Dremel queries return approximate results (e.g. top-k, join)
Outline

• Introduction
• Nested Columnar Storage
• Query Processing
• Experiments and Observations
Record v.s. Columns

Tablet: 375 MB (compressed), 300K rows, 125 columns

(a) read + decompress
(b) assemble records
(c) parse as C++ objects
(d) read + decompress
(e) parse as C++ objects
Record v.s. Columns: Takeaways

• For columnar storage, the most significant performance gain occurs when few fields (columns) are read

• Record assembly and parsing are expensive

• Even when we need records, it is still better to store data in columnar format

• Record-based storage gradually start to outperform Columnar storage if more fields are read
Map Reduce v.s. Dremel

Execution time (sec) on 3000 nodes, 85 billion records
Map Reduce v.s. Dremel: Sidenote

- Dremel is not designed to replace Map Reduce. Rather, it is used in conjunction with Map Reduce.

- Map Reduce is a **generic software framework** designed to tackle **distributed computational** problems for large data

- Dremel is a **data analysis tool** that runs almost **realtime**

- The two were designed with different purposes.
Map Reduce v.s. Dremel: Sidenote

• Why do we need Dremel? Why not just Map Reduce?

• Map Reduce and the other frameworks built on top of it (e.g. Hive, Pig) have a latency between running the job and getting the answer. In other words, they are not realtime.

• Dremel complements that weakness.
Scalability

![Scalability Graph]

- **x-axis**: number of leaf servers
- **y-axis**: execution time (sec)

The graph shows a decreasing trend in execution time as the number of leaf servers increases, indicating improved scalability.
Observations

- Dremel scans quadrillions of records per month
- Most queries are processed under 10 seconds
- Map Reduce can benefit from Columnar Storage just like a DBMS
- Parallel DBMS can benefit from serving tree architecture just like search engines
- Possible to analyze large disk-resident datasets interactively on basic hardware
  - 1T records, thousands of nodes
Recap
Dremel

• A distributed system for interactive analysis of large datasets
  • Thousands of nodes, Petabytes of data
  • Returns answers in seconds
  • Read-only data

• Nested data model
  • Thousands of fields, deeply nested

• Columnar storage
  • Much faster than record-oriented storage in reading time
  • Lossless representation of record structure

• Serving tree architecture
  • Aggregation of results and query scheduling in parallel
Thank you!
Q&A