Dremel: Interactive Analysis of Web-Scale Datasets

S. Melnik, A. Gubarev, J. Long, G. Romer,
S. Shivakumar, M. Tolton
Google Inc.
VLDB 2010
Presented by Ke Hong
(slide adapted from Melnik’s)
Outline

- Problem
- Motivation
- Nested Columnar Storage
- Query Execution
- Evaluation
- Summary
Interactive Query

- Run a MapReduce to extract billions of signals from web pages
- Launch Dremel to execute several interactive commands and extract an irregularity in $signal1$

```sql
DEFINE TABLE t AS /path/to/data/*
SELECT TOP(signal1, 100), COUNT(*) FROM t
...
```

- Feed the incoming input data into another MapReduce pipeline
Problem

- How to do SQL-like aggregation queries
  - On trillion-row tables
  - On petabytes of data
  - Using thousands of CPUs
  - Executed within seconds
  - Serving thousands of concurrent users
Motivation

- Analysis of crawled web documents
- Tracking install data for applications on Android Market
- Crash reporting for Google products
- Nested Columnar Storage
- 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
Motivation

- Parallel DBMS
  - Limited scalability
  - Limited interactive speed
  - Not fault tolerant

- Translating SQL into MapReduce jobs
  - HadoopDB, Pig Latin, Hive
  - One job takes thousands of seconds
  - Hard to support multi-user
Dremel Data Layout

- Nested structure

\[ \tau = \text{dom} \mid <A_1:\tau[*|?], \ldots, A_n:\tau[*|?] > \]

\( \tau \) as an atomic or record type

* as repeated fields

? as optional fields

- Row-oriented: \( r_1 \) and \( r_2 \)

- Column-oriented: \( r_1 \) and \( r_2 \)

Read less, cheaper decompression
Row vs. Column

Execution time on 3000 nodes to process an 85 billion record table.
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>value</td>
<td>value</td>
</tr>
<tr>
<td></td>
<td>r</td>
<td>d</td>
<td>r</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>20</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>
<tr>
<td></td>
<td><a href="http://C">http://C</a></td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>20</td>
<td>20</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>40</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>60</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>80</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>NULL</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>30</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name.Language.Code</th>
<th>Name.Language.Country</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>r</td>
</tr>
<tr>
<td>en-us</td>
<td>0</td>
</tr>
<tr>
<td>en</td>
<td>2</td>
</tr>
<tr>
<td>NULL</td>
<td>1</td>
</tr>
<tr>
<td>en-gb</td>
<td>1</td>
</tr>
<tr>
<td>NULL</td>
<td>0</td>
</tr>
</tbody>
</table>
Repetition and definition levels encode the structural *delta* between the current value and the previous value.

- **r**: length of common path prefix
- **d**: length of the current path

<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>
<tr>
<td>en</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NULL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>en-gb</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NULL</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Example

- **r**:
  - r₁.Name₁.Language₁.Code: 'en-us'
  - r₁.Name₁.Language₂.Code: 'en'
  - r₁.Name₂
  - r₁.Name₃.Language₁.Code: 'en-gb'
  - r₂.Name₁

---

Repetition & Definition Level
Repetition and definition levels encode the structural *delta* between the current value and the previous value.

- **r**: length of common path prefix
- **d**: length of the current path
Repetition and definition levels encode the structural *delta* between the current value and the previous value.

- **r**: length of common path prefix
- **d**: length of the current path
Repetition and definition levels encode the structural *delta* between the current value and the previous value.

- **r**: length of common path prefix
- **d**: length of the current path

### Table: Name.Language.Code

<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></td>
<td></td>
</tr>
</tbody>
</table>

### Example

- \( r_1.\text{Name}_1.\text{Language}_1.\text{Code}: 'en-us' \)
- \( r_1.\text{Name}_1.\text{Language}_2.\text{Code}: 'en' \)
- \( r_1.\text{Name}_2 \)
- \( r_1.\text{Name}_3.\text{Language}_1.\text{Code}: 'en-gb' \)
- \( r_2.\text{Name}_1 \)
Repetition and definition levels encode the structural \textit{delta} between the current value and the previous value.

\textbf{r}: length of common path prefix

\textbf{d}: length of the current path

<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>

\[ r_1.\text{Name}_1.\text{Language}_1.\text{Code}: 'en-us' \]
\[ r_1.\text{Name}_1.\text{Language}_2.\text{Code}: 'en' \]
\[ r_1.\text{Name}_2 \]
\[ r_1.\text{Name}_3.\text{Language}_1.\text{Code}: 'en-gb' \]
\[ r_2.\text{Name}_1 \]
Record Assembly

Transitions labeled with repetition levels
Record Assembly

- Simpler FSM to retrieve a subset of fields
- Cheaper to execute
- Useful for queries like

/Name[2]/Language[1]/Country

DocId: 10
Name
Language
Country: 'us'
Language
Name
Name
Language
Country: 'gb'

DocId: 20
Name
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>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Cnt: 2</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Language</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Str: '<a href="http://A,en-us">http://A,en-us</a>'</td>
</tr>
<tr>
<td>Str: '<a href="http://A,en">http://A,en</a>'</td>
</tr>
<tr>
<td>Name</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<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;
    }
  }
}
```
Multi-Level Serving Tree

- Parallelize scheduling and aggregation
- Mostly one-pass aggregation
- Well suited for aggregation queries returning small and medium-sized results (<1M records)
- Common class of interactive queries

client

root server

intermediate servers

leaf servers (with local storage)

storage layer (e.g., GFS)
Multi-Level Serving Tree

- Receive a query from client, determine all tablets
Multi-Level Serving Tree

- Rewrite the query and route it to intermediate servers
Multi-Level Serving Tree

- Rewrite the query into a set of partial queries and assign them to leaf servers
Multi-Level Serving Tree

- Access assigned tablets from the storage layer

client

root server

intermediate servers

leaf servers (with local storage)

storage layer (e.g., GFS)
Example: count()

\[ \text{SELECT A, COUNT(B) FROM T} \]
\[ \quad \text{GROUP BY A} \]
\[ T = \{ /gfs/1, /gfs/2, \ldots, /gfs/100000 \} \]

\[ \text{SELECT A, SUM(c) FROM (} R_1^1 \cup \ldots \cup R_n^1 \text{)} \]
\[ \quad \text{GROUP BY A} \]
Example: count()

0

SELECT A, COUNT(B) FROM T
GROUP BY A
T = {/gfs/1, /gfs/2, …, /gfs/100000}

SELECT A, SUM(c)
FROM (R^1_1 \cup \ldots \cup R^1_n)
GROUP BY A

1

R^1_1

SELECT A, COUNT(B) AS c
FROM T^1_1
GROUP BY A
T^1_1 = {/gfs/1, …, /gfs/10000}

R^1_2

SELECT A, COUNT(B) AS c
FROM T^1_2
GROUP BY A
T^1_2 = {/gfs/10001, …, /gfs/20000}

\ldots
Example: count()

SELECT A, COUNT(B) FROM T
GROUP BY A
T = {/gfs/1, /gfs/2, …, /gfs/100000}

SELECT A, SUM(c) FROM (R^1_1 UNION ALL … R^1_n) GROUP BY A

R^1_1

SELECT A, COUNT(B) AS c FROM T^1_1 GROUP BY A
T^1_1 = {/gfs/1, …, /gfs/100000}

R^1_2

SELECT A, COUNT(B) AS c FROM T^1_2 GROUP BY A
T^1_2 = {/gfs/10001, …, /gfs/20000}

R^1_3

SELECT A, COUNT(B) AS c FROM T^1_3 GROUP BY A
T^1_3 = {/gfs/1}

Data access ops
Multi-Level Serving Tree

- Scan assigned tablets to return partial results to intermediate servers

leaf servers (with local storage)

storage layer (e.g., GFS)
Multi-Level Serving Tree

- Parallel aggregation of partial results

- Intermediate servers

- Leaf servers (with local storage)

- Storage layer (e.g., GFS)
Multi-Level Serving Tree

• Return the aggregate query result to client
Query Dispatcher

- Dremel as a multi-user system
  - Schedule queries based on priorities and load balancing
  - Fault tolerance

- When the number of tablets processed in one query is larger than the number of available execution threads on leaf servers
  - Compute a histogram of tablet processing times
  - Reschedule tablet processing on another server if a tablet takes disproportionately long time
  - Internal execution tree as a physical query execution plan
Evaluation

- Scalability of Dremel

On a trillion-row table T4 (105TB, 50 fields):

```sql
SELECT TOP(aid, 20), COUNT(*) FROM T4
WHERE bid = {value1} AND cid = {value2}
```
Evaluation

- Interactive speed of Dremel

Monthy query workload of one 3000-node Dremel instance

Most queries complete within 10 sec

execution time (sec)
Summary

- Propose a scenario for interactive queries on web-scale datasets
- Design a columnar storage for Dremel’s nested data model
- Develop SQL-like Dremel’s query language
- Design a multi-level serving tree for efficient query execution
- Evaluate Dremel’s scalability and interactive speed using web-scale workloads