Dremel
Interactive Analysis of Web-Scale Datasets

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

Aravindh Bheemaraj
aravinth@cise.ufl.edu
What is the paper about?

• Overview of Dremel system
• Columnar storage format for nested data
• Dremel’s query language and execution
• Execution trees used in web search systems
• Experimental results
What is Dremel?

- System for interactive analysis of data.
- Uses data, sitting on different storage systems.
- Data modeled in a columnar, semi-structured (Protocol Buffers) format
- Offers SQL-like Query language
Example: Data Exploration

- Runs a MapReduce to extract billions of signals from web pages

- Ad hoc SQL against Dremel
  
  DEFINE TABLE t AS /path/to/data/*
  
  SELECT TOP(signal, 100), COUNT(*) FROM t

- More MR-based processing on the data (FlumeJava, Sawzall)

- Can register the new dataset in a project
Dremel system

- Trillion-record, multi-terabyte datasets at interactive speed
  - Scales to thousands of nodes
  - Fault tolerant execution
- Nested data model
  - Complex datasets; normalization is prohibitive
  - Columnar storage and processing
- Tree architecture (as in web search)
- Interoperates with Google's data mgmt tools
  - In situ data access (e.g., GFS, Bigtable)
Widely used 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
What makes Dremel powerful?

• Common Storage Layer – Ex: GFS
  – In-situ data management
  – Data can be conveniently manipulated
    • Transfer to other clusters
    • Change access privileges

• Shared Storage Format
  – Columnar storage
Record-wise vs. Columnar Representation of Nested data

- Storage size?
- How much data is read?
- How to preserve structure?
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-stripped representation

<table>
<thead>
<tr>
<th>Docld</th>
<th>Name.Url</th>
<th>Links.Forward</th>
<th>Links.Backward</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>value</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>0</td>
</tr>
<tr>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><a href="http://A">http://A</a></td>
<td>0</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td><a href="http://B">http://B</a></td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>NULL</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td><a href="http://C">http://C</a></td>
<td>0</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>40</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>0</td>
<td>2</td>
<td></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></td>
</tr>
<tr>
<td>30</td>
<td>1</td>
<td>2</td>
<td></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>value</td>
</tr>
<tr>
<td>r</td>
<td>d</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>
<tr>
<td>us</td>
<td>0</td>
</tr>
<tr>
<td>NULL</td>
<td>2</td>
</tr>
<tr>
<td>NULL</td>
<td>1</td>
</tr>
<tr>
<td>gb</td>
<td>1</td>
</tr>
<tr>
<td>NULL</td>
<td>0</td>
</tr>
<tr>
<td>r : representation levels</td>
<td></td>
</tr>
<tr>
<td>d : definition levels</td>
<td></td>
</tr>
</tbody>
</table>
Repetition and definition levels

<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>

r: At what repeated field in the field's path the value has repeated

d: How many fields in paths that could be undefined (opt. or rep.) are actually present
Record assembly FSM

Figure 6. Complete record assembly automaton. Edges are labeled with repetition levels.
Query processing

• Tree architecture
  – Leaf nodes access data
  – Higher level nodes do parallel aggregation

• Optimized for select-project-aggregate
  – Very common class of interactive queries
  – Single scan
  – Within-record and cross-record aggregation

• Approximations: count(distinct), top-k
  – minimum % of tablets that can be read
SQL dialect for nested data

```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</th>
<th>Name</th>
<th>Cnt</th>
<th>Language</th>
<th>Str</th>
<th>Str</th>
</tr>
</thead>
</table>

Output schema

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

No record assembly during query processing
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 UNION ALL R_110)
GROUP BY A
```

```
SELECT A, COUNT(B) AS c
FROM T_1
GROUP BY A
```

```
SELECT A, COUNT(B) AS c
FROM T_11
GROUP BY A
```

```
SELECT A, COUNT(B) AS c
FROM T_12
GROUP BY A
```

```
SELECT A, COUNT(B) AS c
FROM T_13
GROUP BY A
```

```
R_1
```

```
SELECT A, COUNT(B) AS c
FROM T_1
GROUP BY A
```

```
R_12
```

```
SELECT A, COUNT(B) AS c
FROM T_12
GROUP BY A
```

```
SELECT A, COUNT(B) AS c
FROM T_13
GROUP BY A
```

```
T_1 = {/gfs/1, ..., /gfs/100000}
```

```
T_11 = {/gfs/1, ..., /gfs/100000}
```

```
T_12 = {/gfs/10001, ..., /gfs/20000}
```

```
T_13 = {/gfs/1}
```

```
Data access ops
```

```
...  ...
```

```
...  ...
```

```
...  ...
```

```
...  ...
```

```
...  ...
```
Read from disk

"cold" time on local disk, averaged over 30 runs

10x speedup using columnar storage

2-4x overhead of using records

Table partition: 375 MB (compressed), 300K rows

(a) read + decompress
(b) assemble records
(c) parse as objects
(d) read + decompress
(e) parse as objects
MR and Dremel execution

Avg # of terms in txtField in 85 billion record table T1

Sawzall program ran on MR:

num_recs: table sum of int;
num_words: table sum of int;
emit num_recs <- 1;
emit num_words <- count_words(input.txtField);

Q1: SELECT SUM(count_words(txtField)) / COUNT(*)
    FROM T1

MR overheads: launch jobs, schedule 0.5M tasks, assemble records
Impact of serving tree depth

Q2:
SELECT country, SUM(item.amount) FROM T2
GROUP BY country

Q3:
SELECT domain, SUM(item.amount) FROM T2
WHERE domain CONTAINS '.net'
GROUP BY domain

40 billion nested items
Q5 on a trillion-row table T4:

```
SELECT TOP(aid, 20), COUNT(*) FROM T4
```
Observations

• Possible to analyze large disk-resident datasets interactively on commodity hardware
  – 1T records, 1000s of nodes

• MR can benefit from columnar storage just like a parallel DBMS
  – But record assembly is expensive
  – Interactive SQL and MR can be complementary

• Parallel DBMSes may benefit from serving tree architecture just like search engines

• Getting to last few percent of data fast is hard
  – Replication, approximation, early termination
Parallel DB + MR (Greenplum) Comparison

Greenplum Demonstration Architecture

- ETL Host
- Client
- Greenplum Data Computing Appliance
- Master
- Standby Master
- Segment
- Private Network
- Public Network

EMC²

© Copyright 2011 EMC Corporation. All rights reserved.
Big questions??

• So, why Dremel is better than other parallel DBs?

• Can MapReduce itself handle fast, interactive querying?

• Dremel available to external world?

• If available for external world, how can it help the customers?
Related Work

- Parallel DBMS + MR
  - Cloudera, Greenplum

- Commercial DBMS implement nested data in XML format

- Pig: SQL-like language operating on nested data.
Thank you!