# Dremel:
# Interactive Analysis of Web-Scale Datasets

SERGEY MELNIK, ANDREY GUBAREV,JING JING LONG,GEOFFREY ROMER,SHIVA SHIVAKUMAR, MATT TOLTON,THEO VASSILAKIS

PRESENTED BY

DIPANNITA DEY

# Outline

- Problem
- Existing technology
- Dremel
  - Basic features
  - Applications
  - Infrastructure & details
- Experiments
- Evaluations

# Problem: Latency Matters

Interactive Tools

Spam

Trends Detection

Real-time Web Dashboards

Network Optimization

# Existing Technologies

- Map-Reduce
  - Record-oriented data
  - Does not work with data in-situ
  - Suitable for batch-processing
- Pig
- Hive

Inherent **Latency** between submitting query and getting result

# Dremel

- Interactive ad-hoc query system
  - Scales to thousands of nodes
  - **Fault tolerant** and handles **stragglers**
  - **SQL** like query language and **multi-level execution trees**

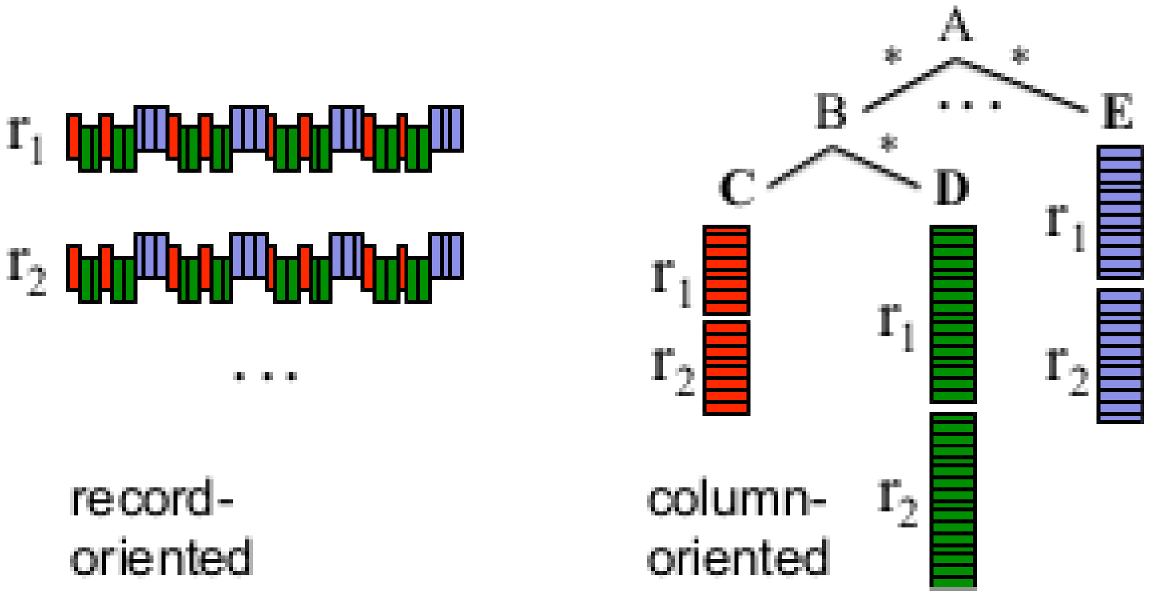
- Nested data model
  - **Columnar storage** of nested (non-relational) data
  - Tree like architecture similar to web search

- Interoperability with data
  - Access data **in situ** (Eg. – GFS, Bigtable)
  - MapReduce Pipelines

# Widely used inside Google since 2010

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

# Columnar data storage format



Advantage: Read less, fast access, lossless representation

Challenge: preserve structure, reconstruct from a subset of fields

# 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'
```

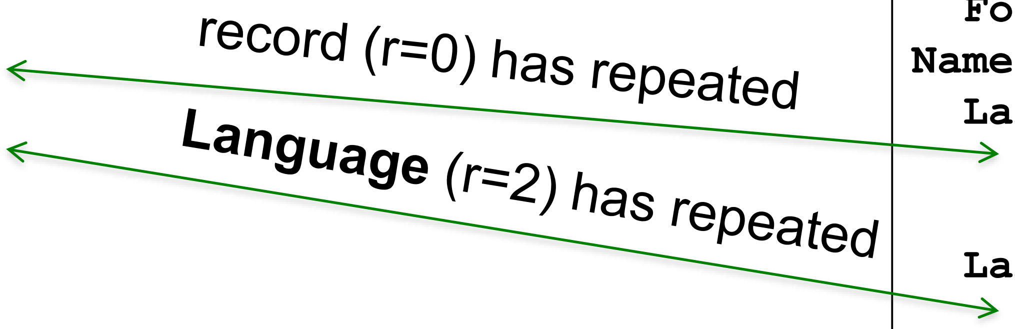
```
DocId: 20
Links
  Backward: 10
  Backward: 30
  Forward:  80
Name
  Url: 'http://C'
```

# Repetition and definition levels

r=1 r=2 (non-repeating)

**Name.Language.Code**

| value | r | d |
|---|---|---|
| en-us | 0 | 2 |
| en | 2 | 2 |
| NULL | 1 | 1 |
| en-gb | 1 | 2 |
| NULL | 0 | 1 |



record (r=0) has repeated

**Language** (r=2) has repeated

```
DocId: 10                r1
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                r2
Links
  Backward: 10
  Backward: 30
  Forward:  80
Name
  Url: 'http://C'
```

**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

# Column-striped representation

**DocId**

| value | r | d |
|---|---|---|
| 10 | 0 | 0 |
| 20 | 0 | 0 |

**Name.Url**

| value | r | d |
|---|---|---|
| http://A | 0 | 2 |
| http://B | 1 | 2 |
| NULL | 1 | 1 |
| http://C | 0 | 2 |

**Links.Forward**

| value | r | d |
|---|---|---|
| 20 | 0 | 2 |
| 40 | 1 | 2 |
| 60 | 1 | 2 |
| 80 | 0 | 2 |

**Links.Backward**

| value | r | d |
|---|---|---|
| NULL | 0 | 1 |
| 10 | 0 | 2 |
| 30 | 1 | 2 |

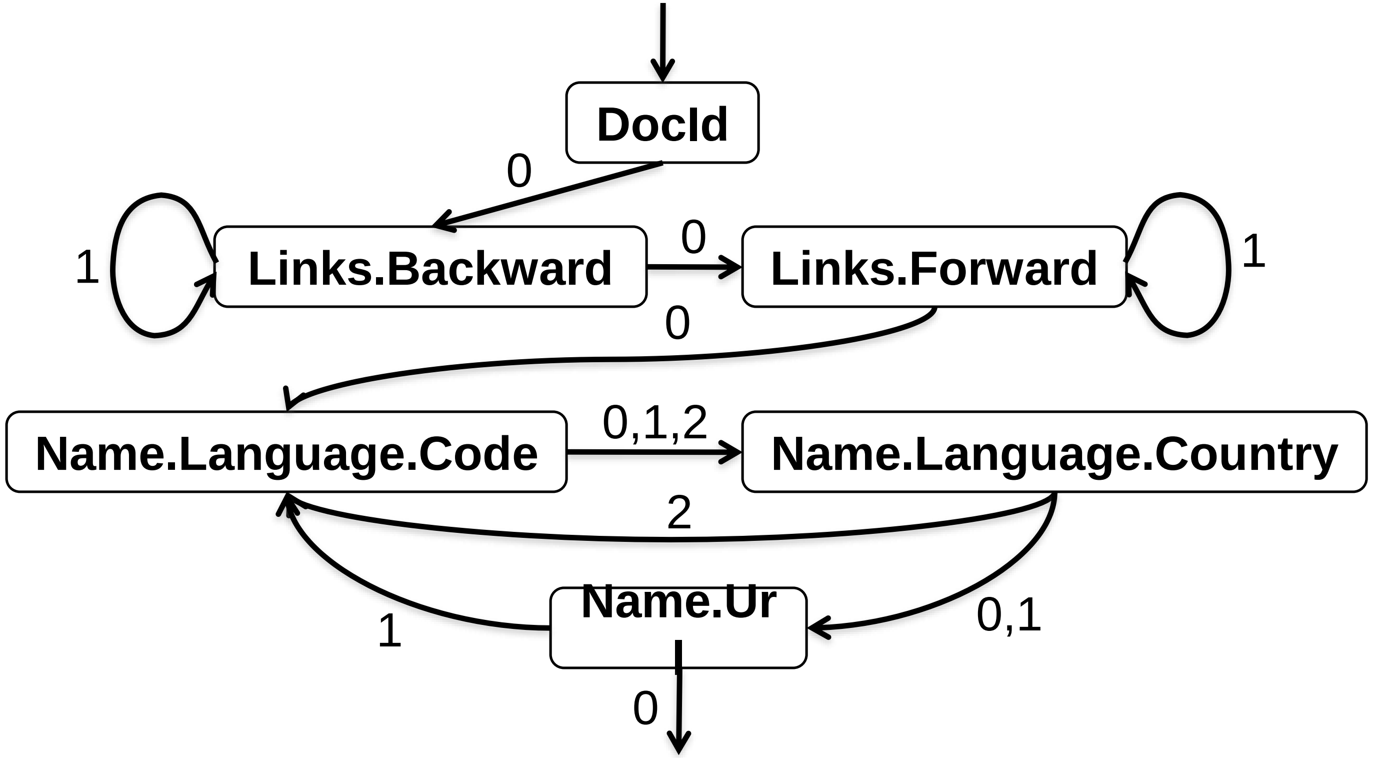
**Name.Language.Code**

| value | r | d |
|---|---|---|
| en-us | 0 | 2 |
| en | 2 | 2 |
| NULL | 1 | 1 |
| en-gb | 1 | 2 |
| NULL | 0 | 1 |

**Name.Language.Country**

| value | r | d |
|---|---|---|
| us | 0 | 3 |
| NULL | 2 | 2 |
| NULL | 1 | 1 |
| gb | 1 | 3 |
| NULL | 0 | 1 |

# Record assembly FSM



Transitions labeled with repetition levels

For record-oriented data processing (e.g., MapReduce)

# SQL dialect for nested data

```
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

$t_1$

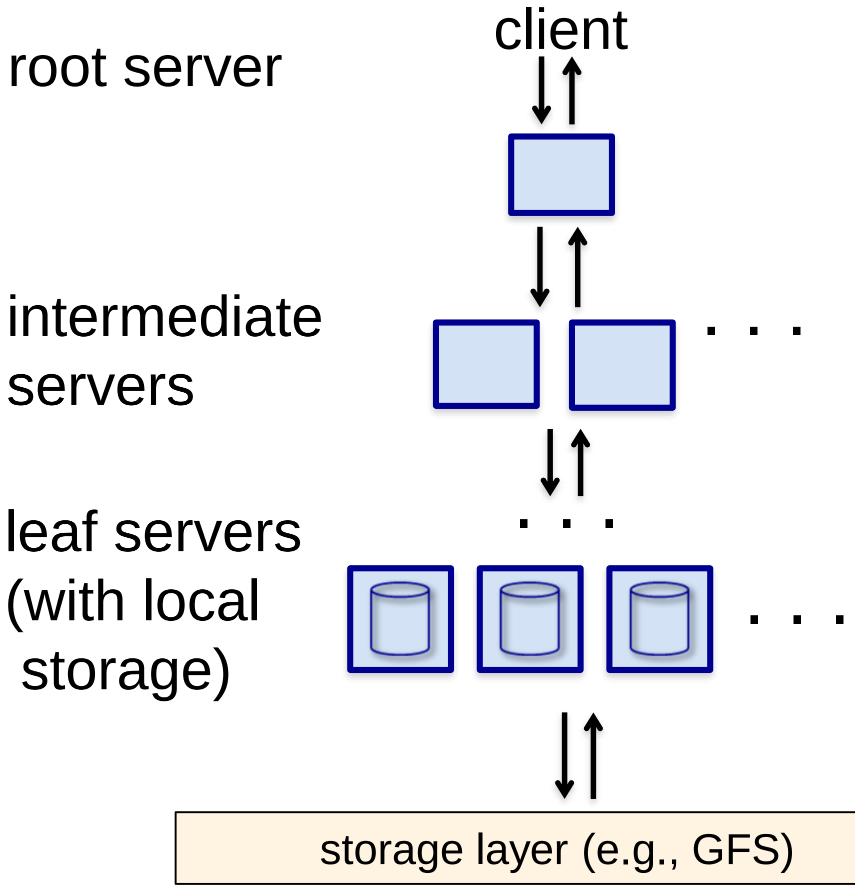
```
Id: 10
Name
  Cnt: 2
  Language
    Str: 'http://A,en-us'
    Str: 'http://A,en'
Name
  Cnt: 0
```

Output schema

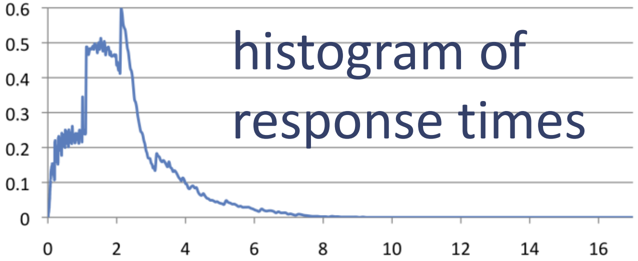
```
message QueryResult {
  required int64 Id;
  repeated group Name {
    optional uint64 Cnt;
    repeated group Language {
      optional string Str;
    }
  }
}
```

No record assembly during query processing

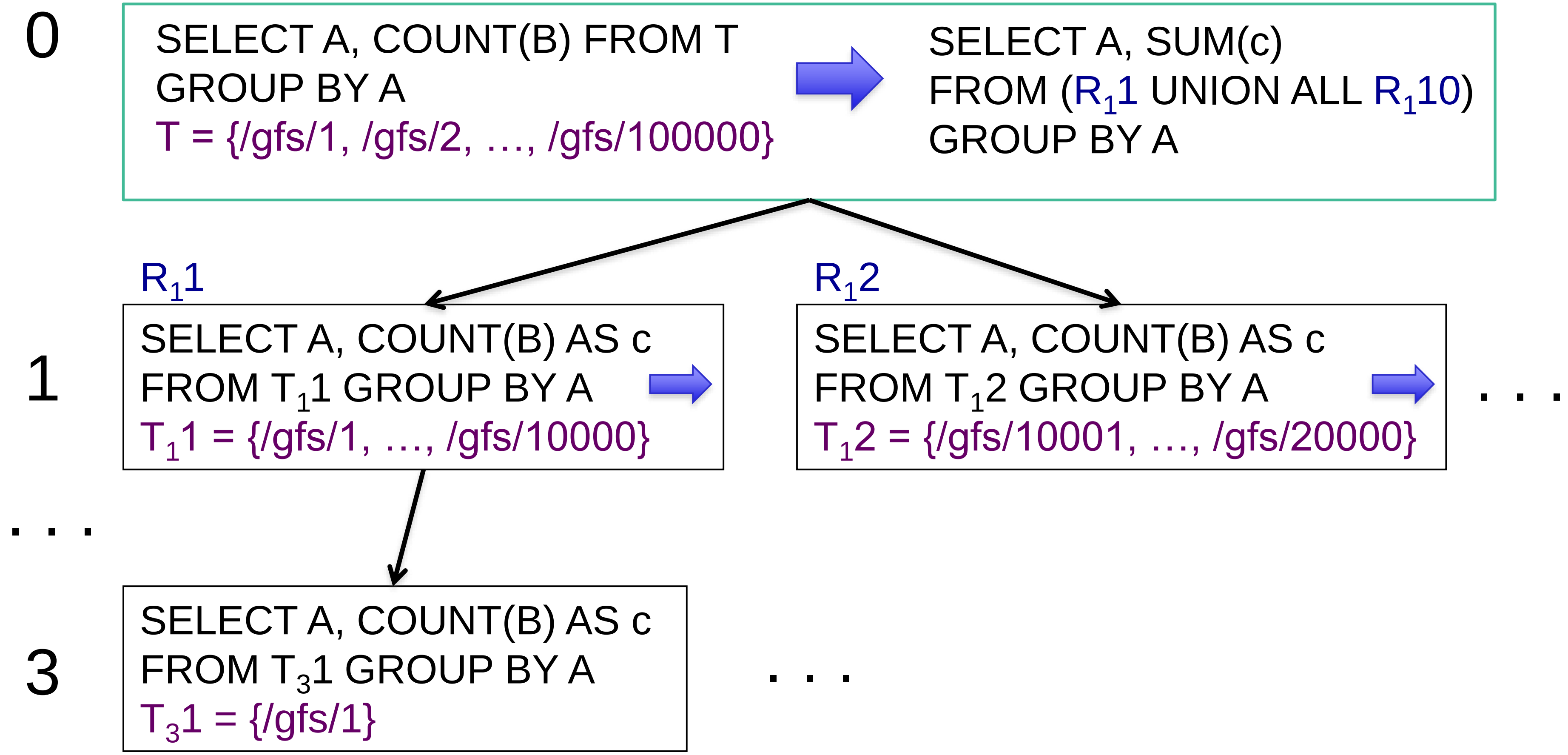
# Serving tree



- Parallelizes scheduling and aggregation
- Fault tolerance
- Stragglers
- Designed for "small" results (<1M records)



# 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 UNION ALL $R_1$10)
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}

. . .

. . .

3

SELECT A, COUNT(B) AS c
FROM $T_3$1 GROUP BY A
$T_3$1 = {/gfs/1}

. . .

Data access ops

# Experiments

- 1 PB of real data
  (uncompressed, non-replicated)
- 100K-800K tablets per table
- Experiments run during business hours

| Table name | Number of records | Size (unrepl., compressed) | Number of fields | Data center | Repl. factor |
|---|---|---|---|---|---|
| T1 | 85 billion | 87 TB | 270 | A | 3× |
| T2 | 24 billion | 13 TB | 530 | A | 3× |
| T3 | 4 billion | 70 TB | 1200 | A | 3× |
| T4 | 1+ trillion | 105 TB | 50 | B | 3× |
| T5 | 1+ trillion | 20 TB | 30 | B | 2× |

# Read from disk

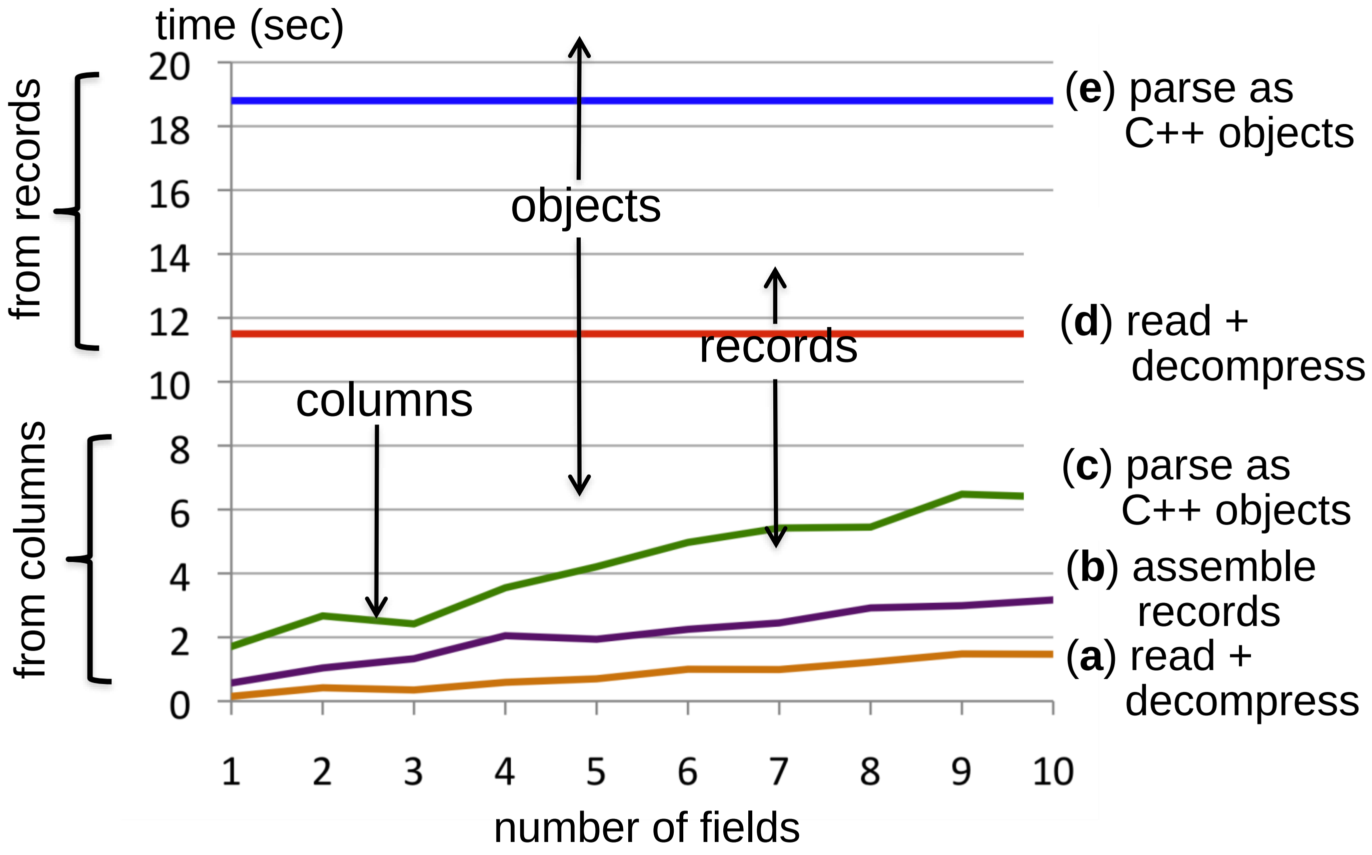
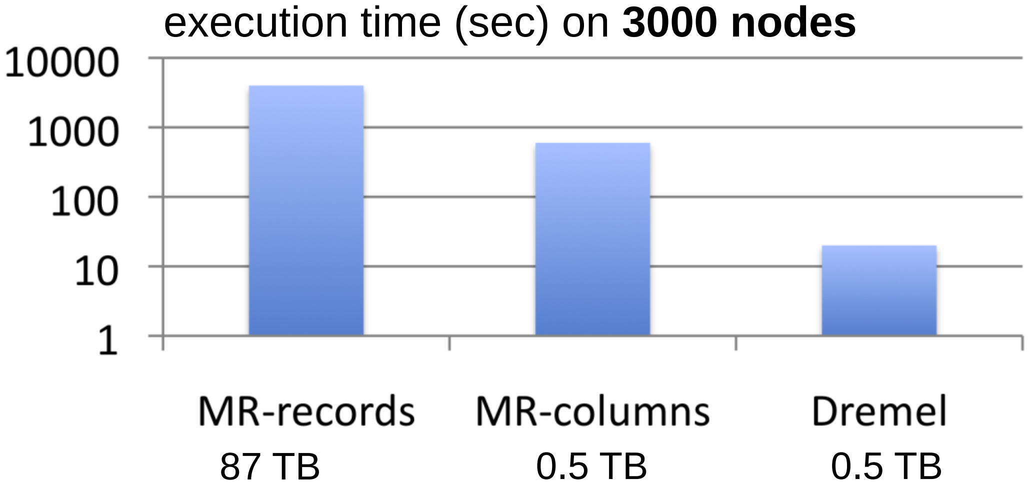
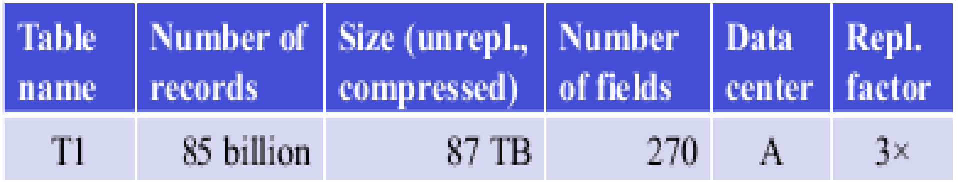


Table partition: 375 MB (compressed), 300K rows, 125 columns

# MR and Dremel execution

Avg # of terms in specific field in table T1



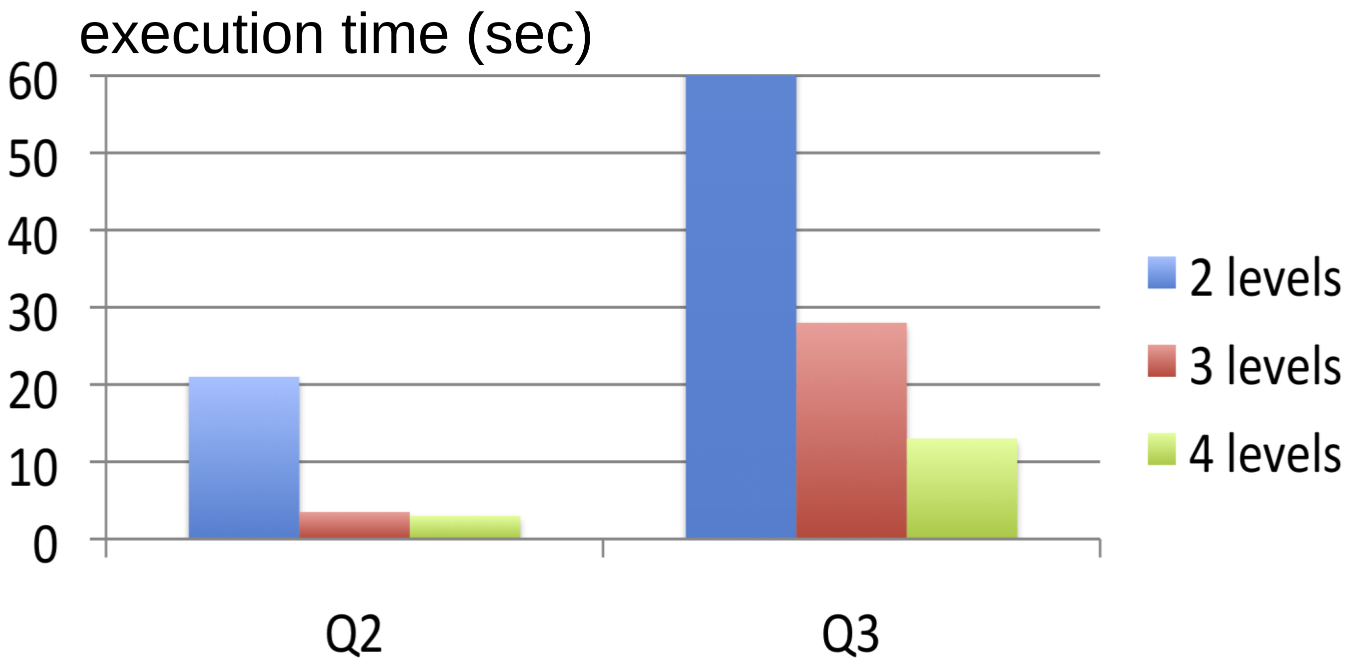


| Table name | Number of records | Size (unrepl., compressed) | Number of fields | Data center | Repl. factor |
|---|---|---|---|---|---|
| T1 | 85 billion | 87 TB | 270 | A | 3× |

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

MR overheads: launch jobs, schedule 0.5M tasks, assemble records

# Impact of serving tree depth



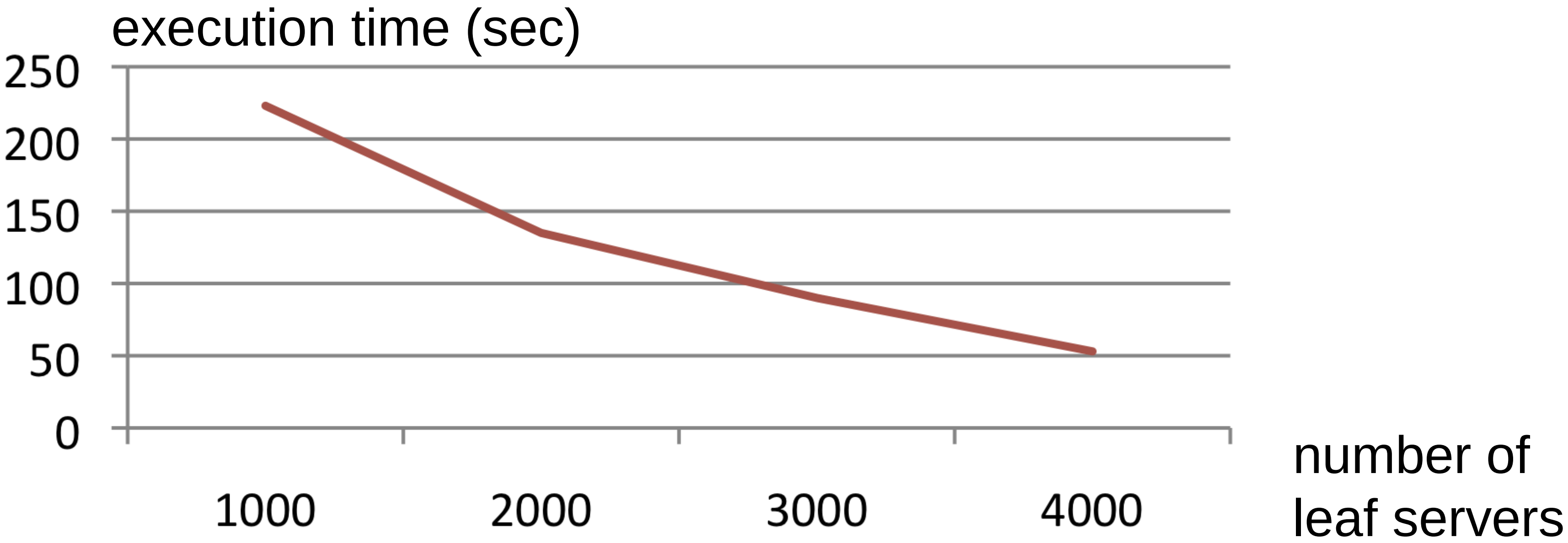
(returns 100s of records)

(returns 1M records)

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
```

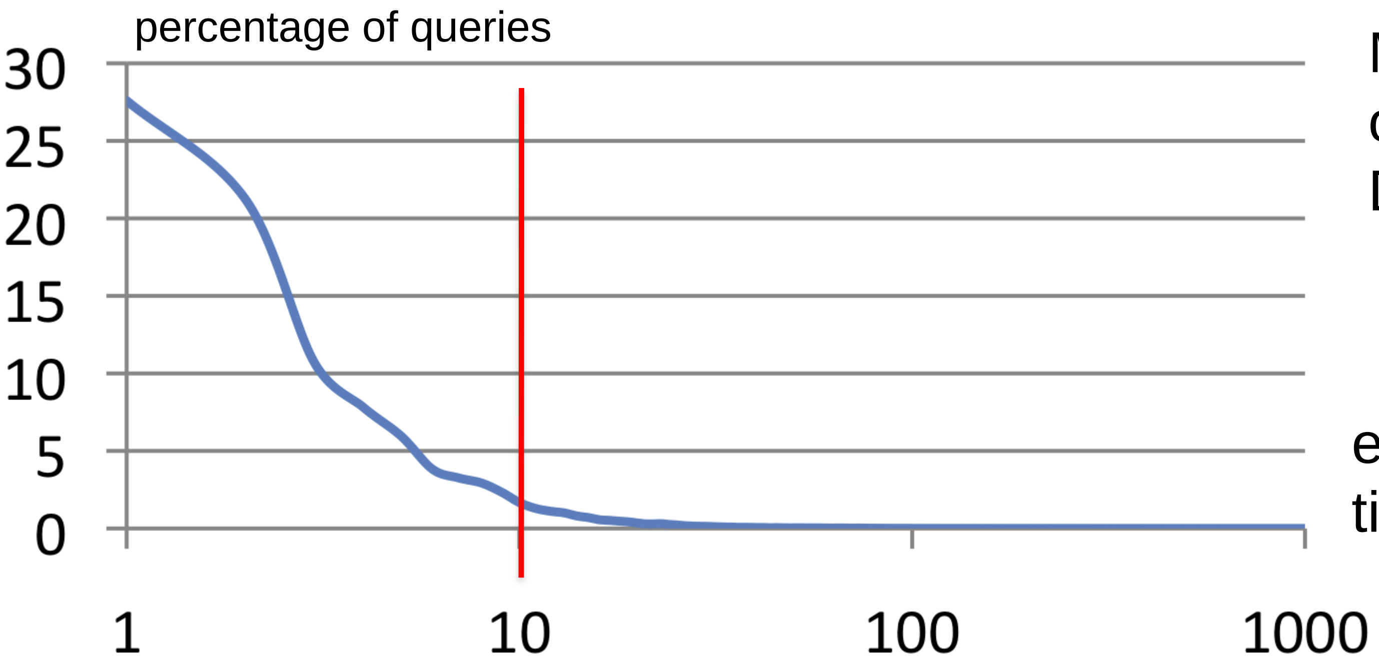
# Scalability



Q5 on a trillion-row table T4:

```
SELECT TOP(aids, 20), COUNT(*) FROM T4
```

# Interactive speed



Monthly query workload of one 3000-node Dremel instance

execution time (sec)

Most queries complete under 10 sec

# Outcome

- Google Big-Query
  - Web Service (pay-per-query)

- Apache Drill
  - Open source Implementation of BigQuery

# Take Away

- Map-Reduce can benefit from columnar storage like a parallel DBMS
  - Record assembly is expensive
  - Dremel complements MR and together produces best results

- Parallel DBMS can benefit from serving tree architecture

# Thank You