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

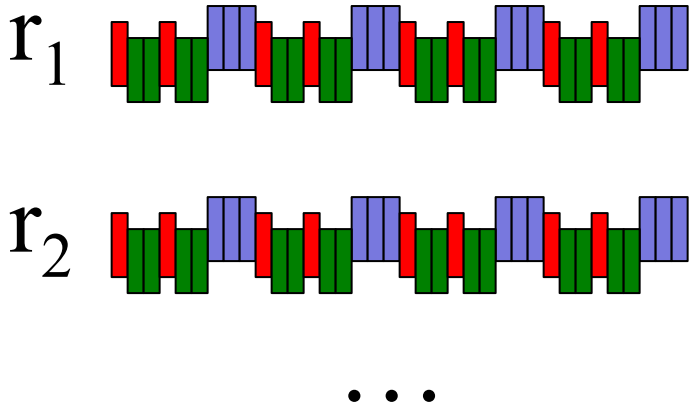
```
DEFINE TABLE t AS /path/to/data/*

SELECT TOP(signal1, 100), COUNT(*) FROM t
```

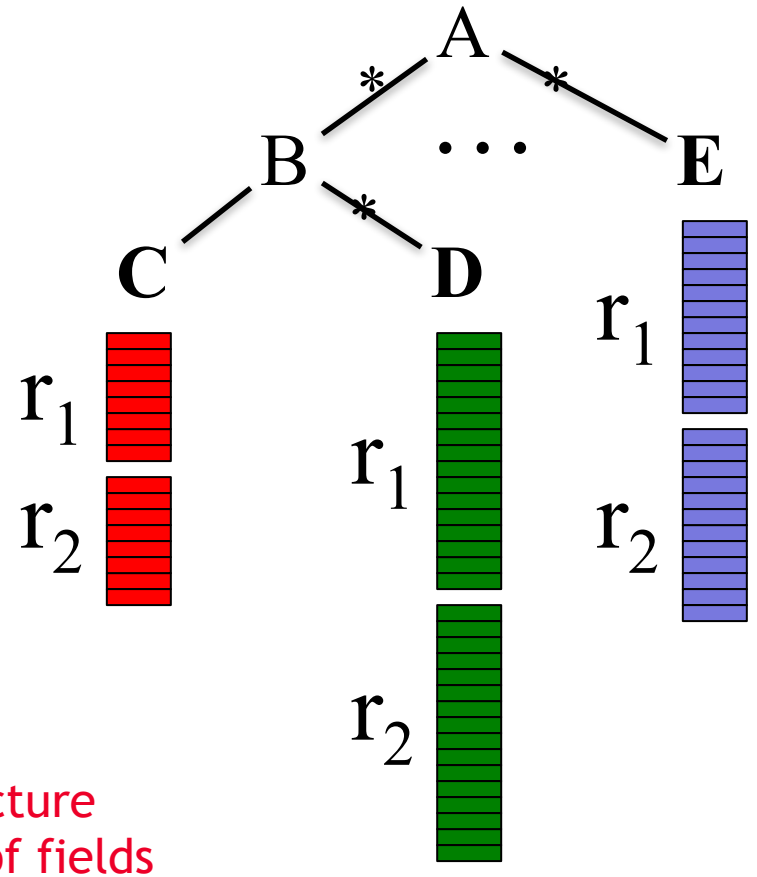
- More MR-based processing of the data

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

multiplicity:

```
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'
```

r_2

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

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

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 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

no value: **Name** (r=1) has repeated, **Name** (d=1) is defined

*no value: record (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'
  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

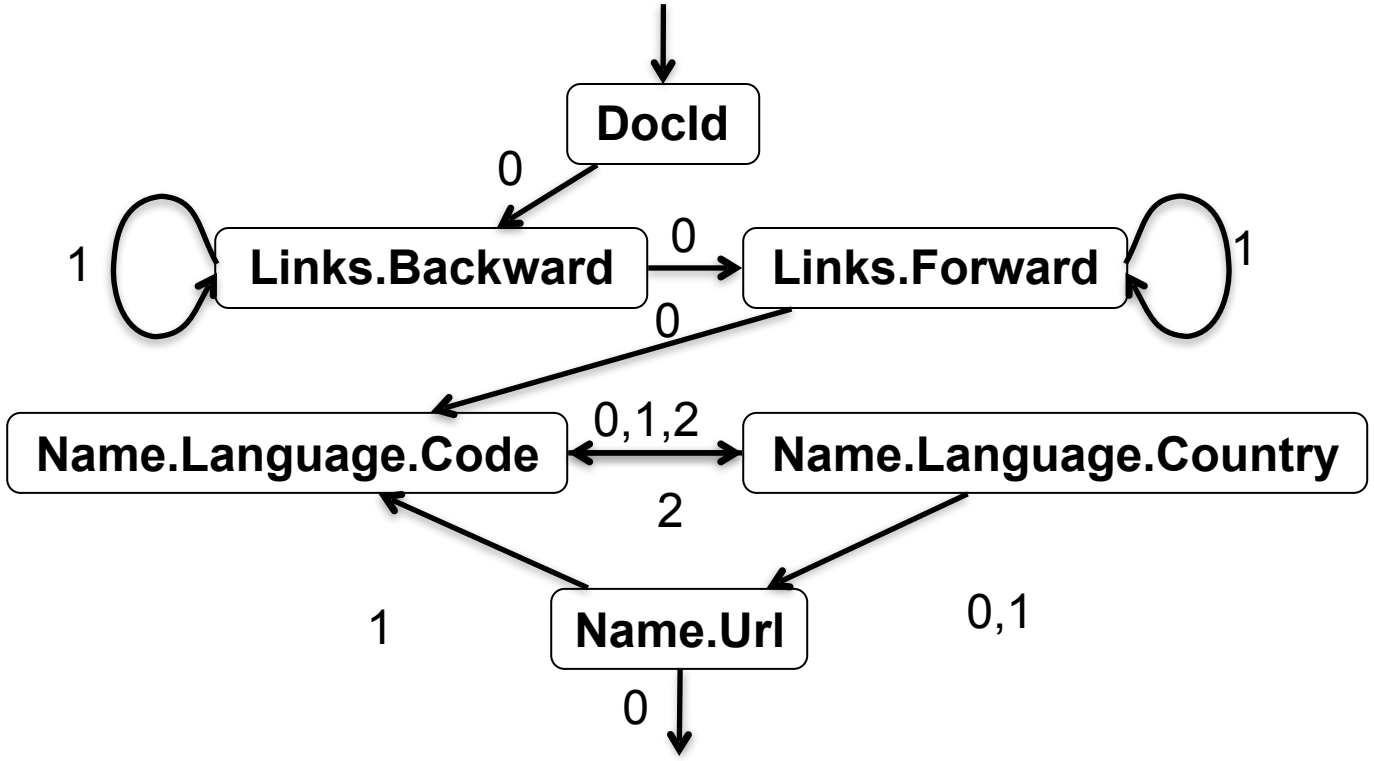
d: How many fields that could be undefined (opt. or rep.) are actually present

Record Assembly

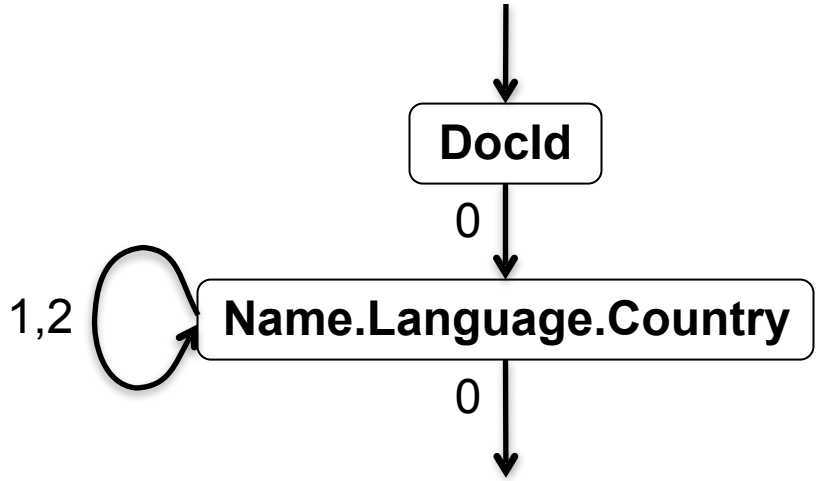
Transitions labeled with repetition levels

- 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



Record Assembly from Two Fields




```
DocId: 10                    s1
Name
  Language
    Country: 'us'
  Language
Name
Name
  Language
    Country: 'gb'
```

```
DocId: 20                    s2
Name
```

Preserves structure of the parent fields

Outline

- Introduction
- Nested Columnar Storage
- Query Processing
- Experiments and Observations

Sample Query

```
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

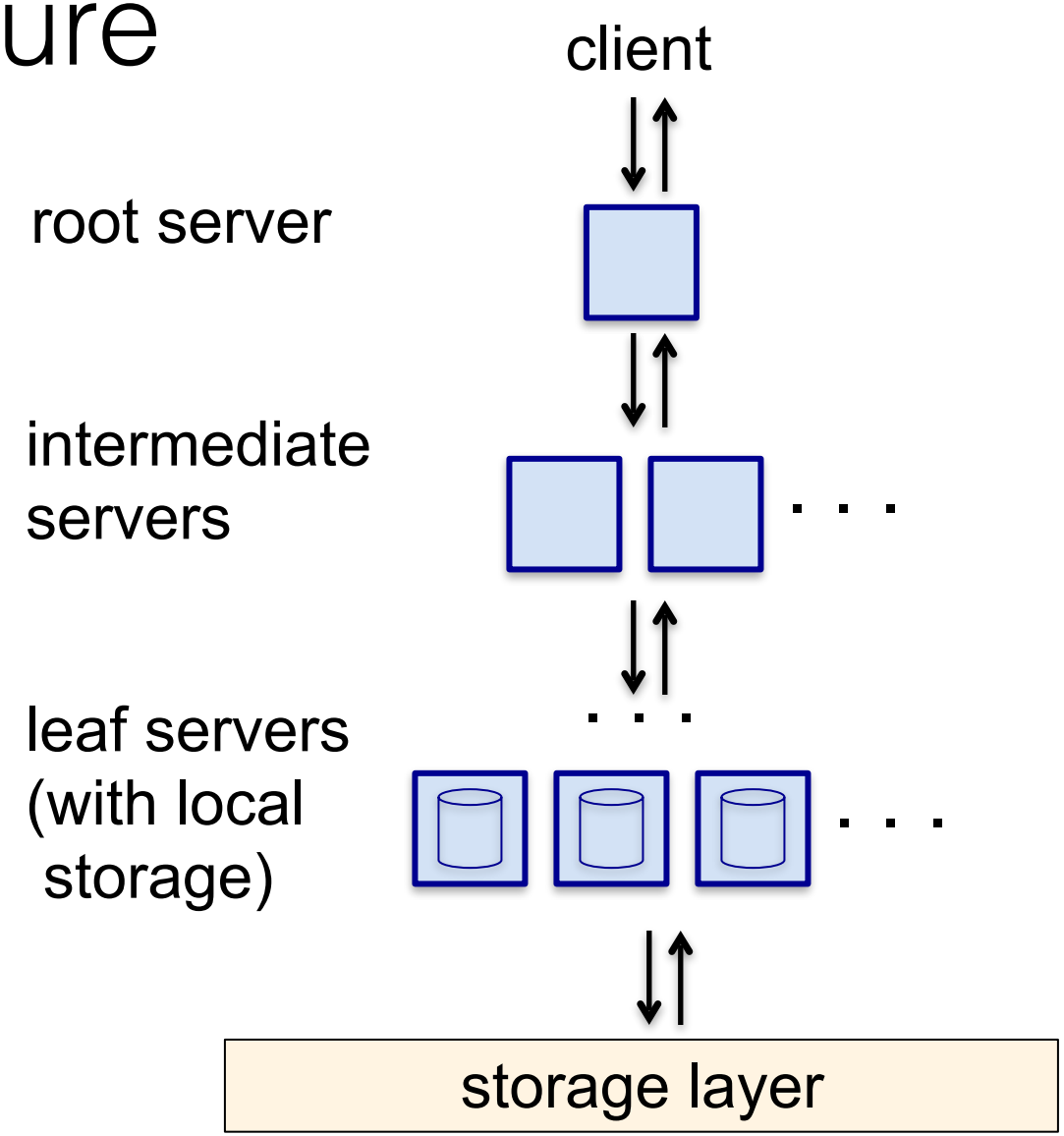
```
Id: 10
Name
  Cnt: 2
  Language
    Str: 'http://A,en-us'
    Str: 'http://A,en'
Name
  Cnt: 0
```

Output schema

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

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

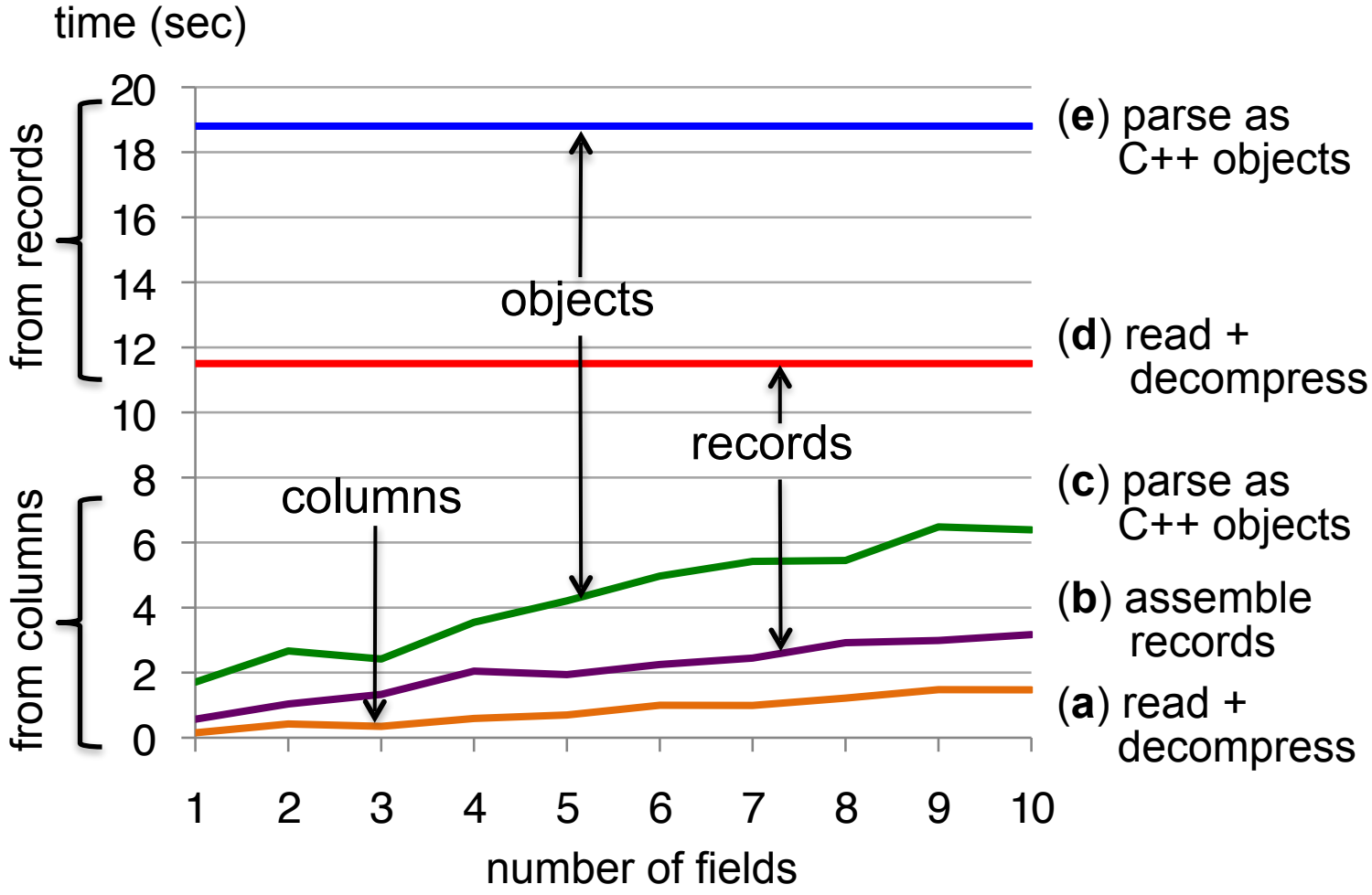
- Query Dispatcher provides scheduling and fault tolerance
  - schedules queries based on their priorities and balances the load
  - If one node becomes much slower, reschedule

- 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

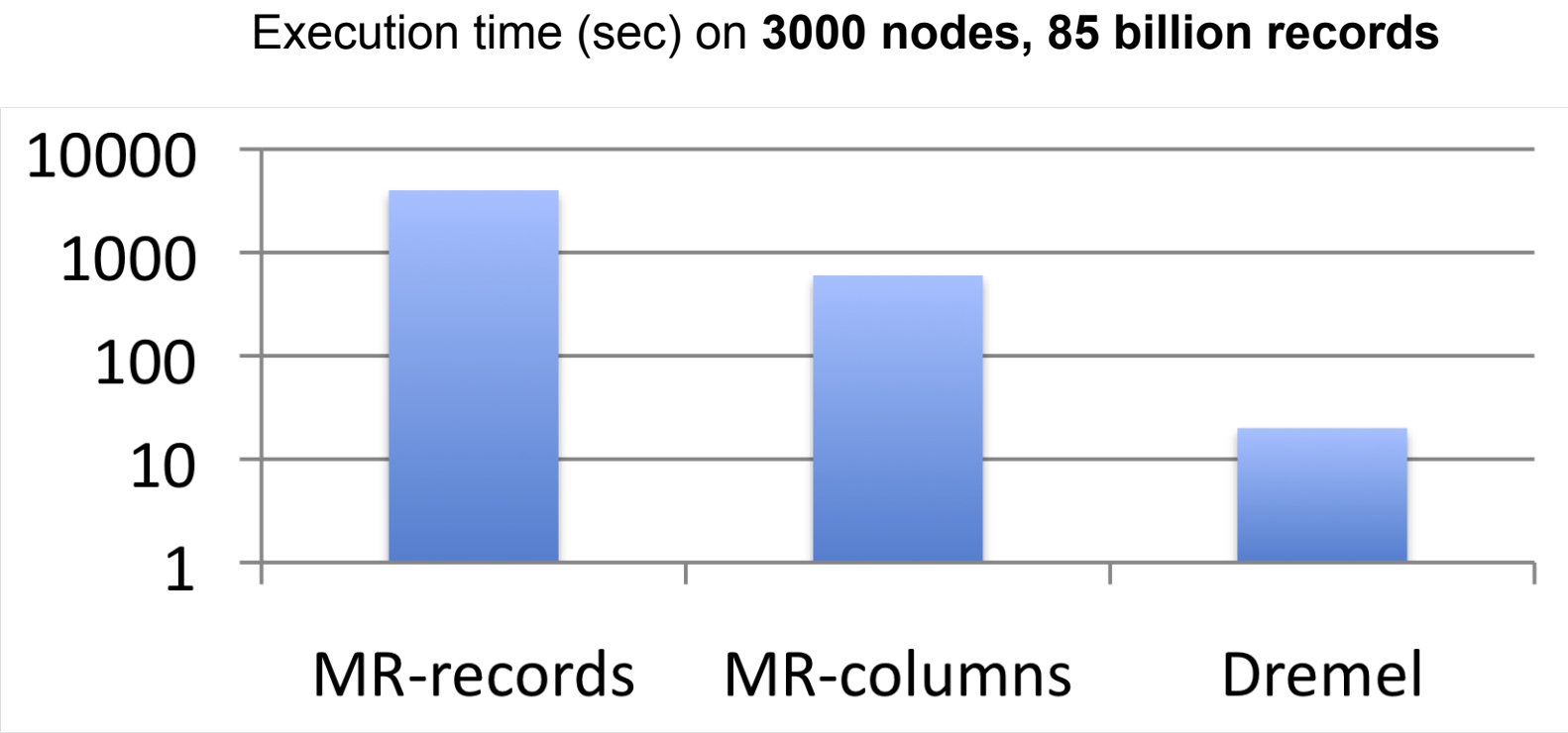


Tablet: 375 MB (compressed), 300K rows, 125 columns

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



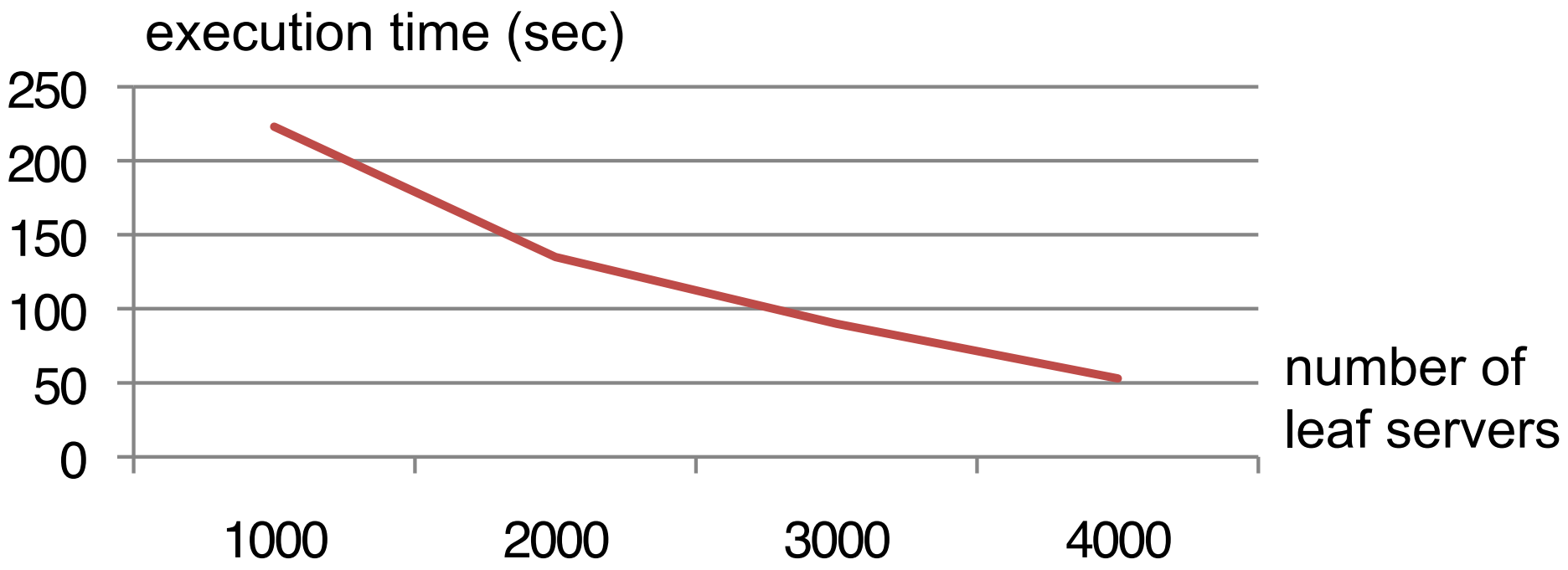
Map Reduce v.s. Dremel: Sidenote

- 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 computation**al 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



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