Dremel: Interactice Analysis of Web-Scale Datasets

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

Presented by: Alex Zahdeh

Overview

- Scalable, **interactive ad-hoc query system** for analysis of **read-only nested data**
- **Multi-level** execution trees, **columnar** data layout
- Capable of aggregation queries over **trillion row** tables in seconds
- Scales to thousands of CPUs and petabytes of data

Motivation

- Need to deal with vast amounts of data spread out over multiple commodity machines
- Interactive queries require **speed**
- Response times make a qualitative difference in many analysis tasks

Applications of Dremel

- 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
- Disk I/O statistics for hundreds of thousands of disks
- Symbols and dependencies in Google's codebase

Data Exploration Example

1. Extract billions of signals from web pages using MapReduce
2. Ad hoc SQL query against Dremel

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

3. More MR based processing

Background

- Requires a common storage layer
  - Google uses GFS
- Requires shared storage format
  - Protocol Buffers

Data Model (Protocol Buffers)

- Nested layout
- Each record consists of one or many data fields
- Fields have a name, type, and multiplicity
- Can specify optional/required fields
- Platform neutral
- Extensible

Data Model Example

```
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'
```

```
message Document {
  required int64 DocId;
  optional group Links {
    repeated int64 Backward;
    repeated int64 Forward; }
  repeated group Name {
    repeated group Language {
      required string Code;
      optional string Country; }
    optional string Url; }}
```

```
DocId: 20                    r2
Links
  Backward: 10
  Backward: 30
  Forward:  80
Name
  Url: 'http://C'
```

Nested Columnar Storage

- Store all values of a given field consecutively
- Improve retreival efficiency
- Challenges
  - Lossless representation of record structure in columnar format
  - Fast encoding and decoding (assembly) of records

Repetition Levels

- Need to disambiguate field repetition and record repetition
- Must store a repetition level to each value

r_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'
```

Definition Levels

- Specifies how many fields that *could* be undefined are actually present in the record
- Stored with each value

Definition Levels Example

```
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'
```

```
message Document {
  required int64 DocId;
  optional group Links {
    repeated int64 Backward;
    repeated int64 Forward; }
  repeated group Name {
    repeated group Language {
      required string Code;
      optional string Country; }
    optional string Url; }}
```

```
DocId: 20                    r2
Links
  Backward: 10
  Backward: 30
  Forward:  80
Name
  Url: 'http://C'
```

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

Encoding

- Each column stored as a set of blocks
- Each block contains:
  - Repetition level
  - Definition level
  - Compressed field values
- NULLS not explicity stored (determined by definition level)

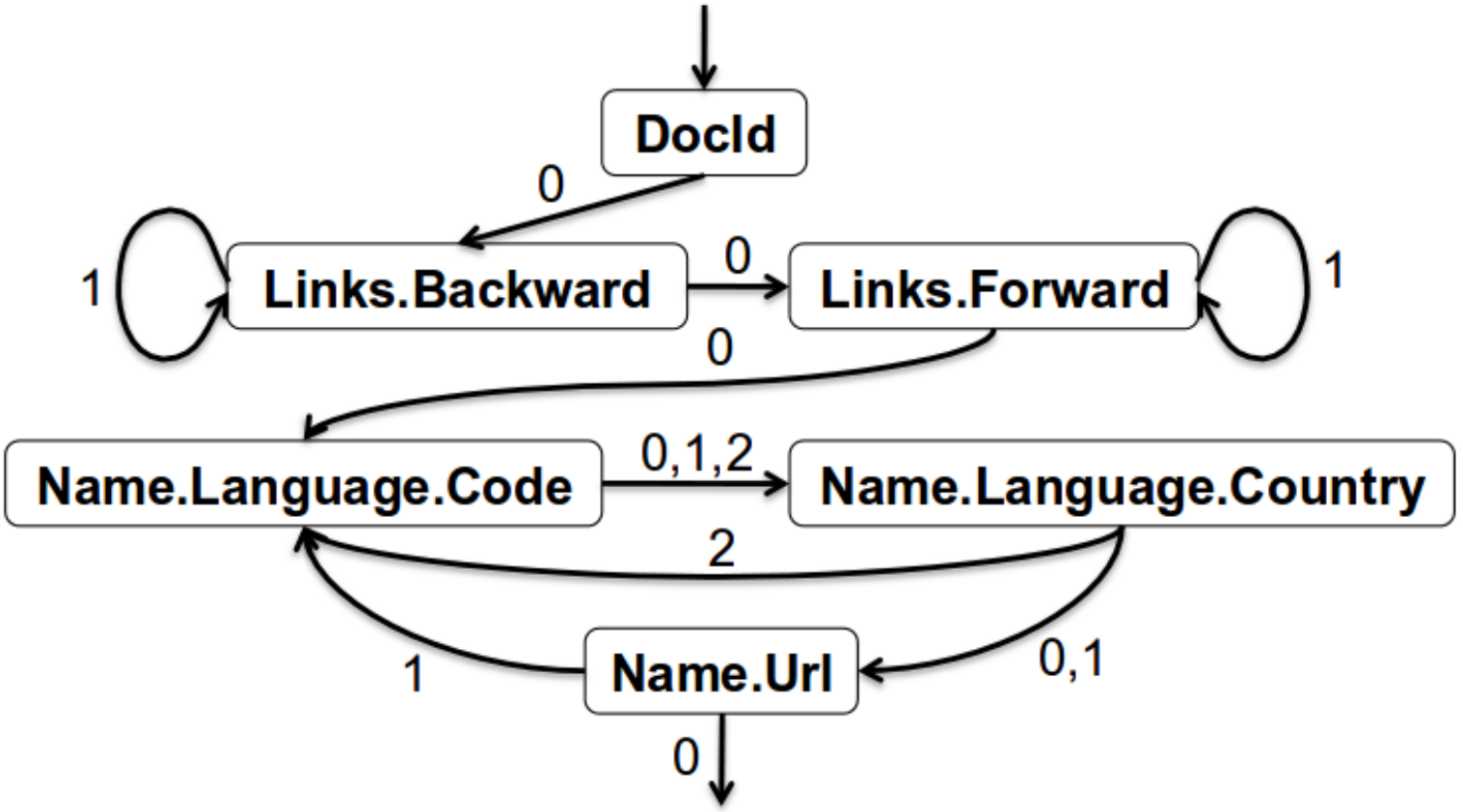
Splitting Records into Columns

- Create a tree of field writers whose structure matches the field heirarchy
- Update field writers only when they have their own data
- Don't propogate state down the tree unless absolutely necessary

Record Assembly

- Finite State Machine that reads the field values and levels and appends the values sequentially to output record
- States correspond to a field reader
- Transitions labeled with repetition levels

Record Assembly FSM



Query Language

- Based on SQL, designed to be efficiently implementable on columnar nested storage
- Each statement takes as input one or more nested tables and their schemas
- Produces a nested table and its output schema

Query Example

```
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;
```

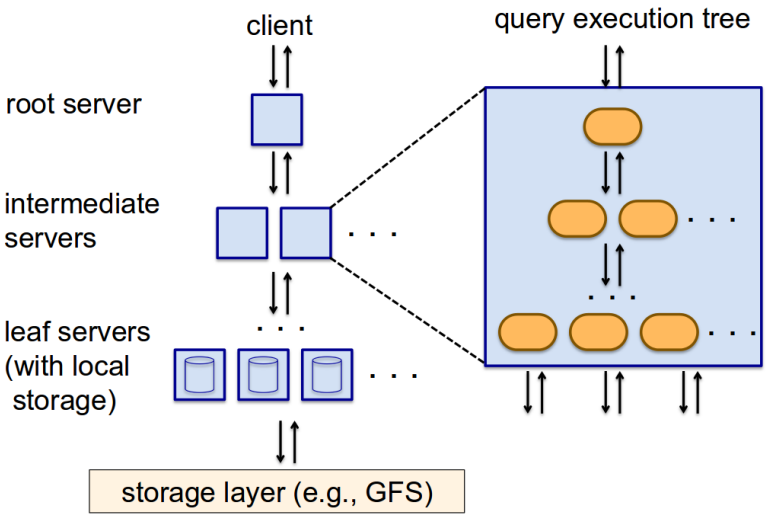
t_1

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

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

Query Execution

- Multi-level serving tree to execute queries
- Partitions of table spread out across leaf servers
- Queries aggregated on the way up
- Designed for "small" results (<1M records)



Query Dispatcher

- Fault tolerance
- Job scheduling
  - *Slots* are available execution threads on leaf servers
  - Amount of data processed larger than number of slots
- Straggler tolerance
  - Redispatch work that is taking too long

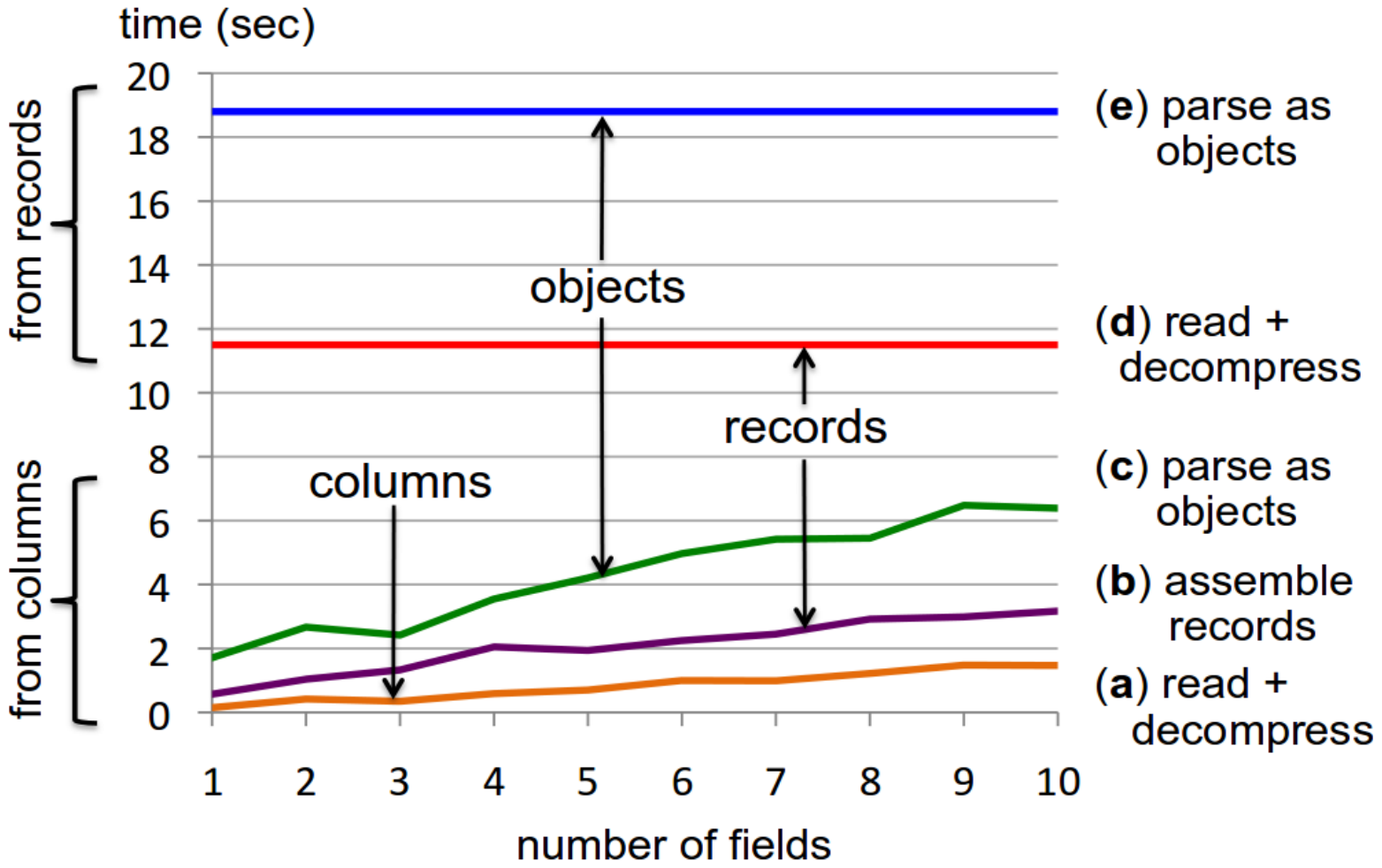
Experiments

- Several datsets
- All tables three way replicated
- Contain from 100k to 800k tablets of various sizes
- Goals
  - Examine access characteristics on a single machine
  - Show benefits of columnar storage for MR execution
  - Show Dremel's performance

Datasets

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×

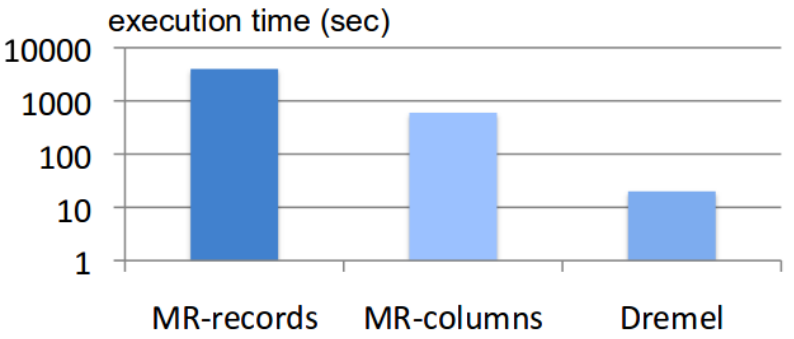
Record vs Column Storage



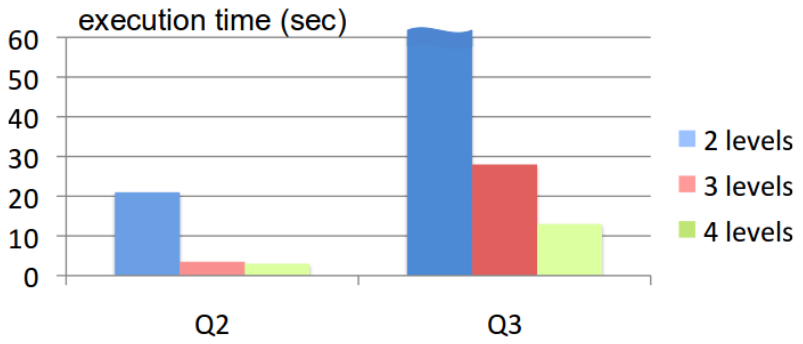
300k record fragment of Table T1 (1GB) used

MR vs Dremel (for aggregation queries)

- Single field access
- 3000 workers



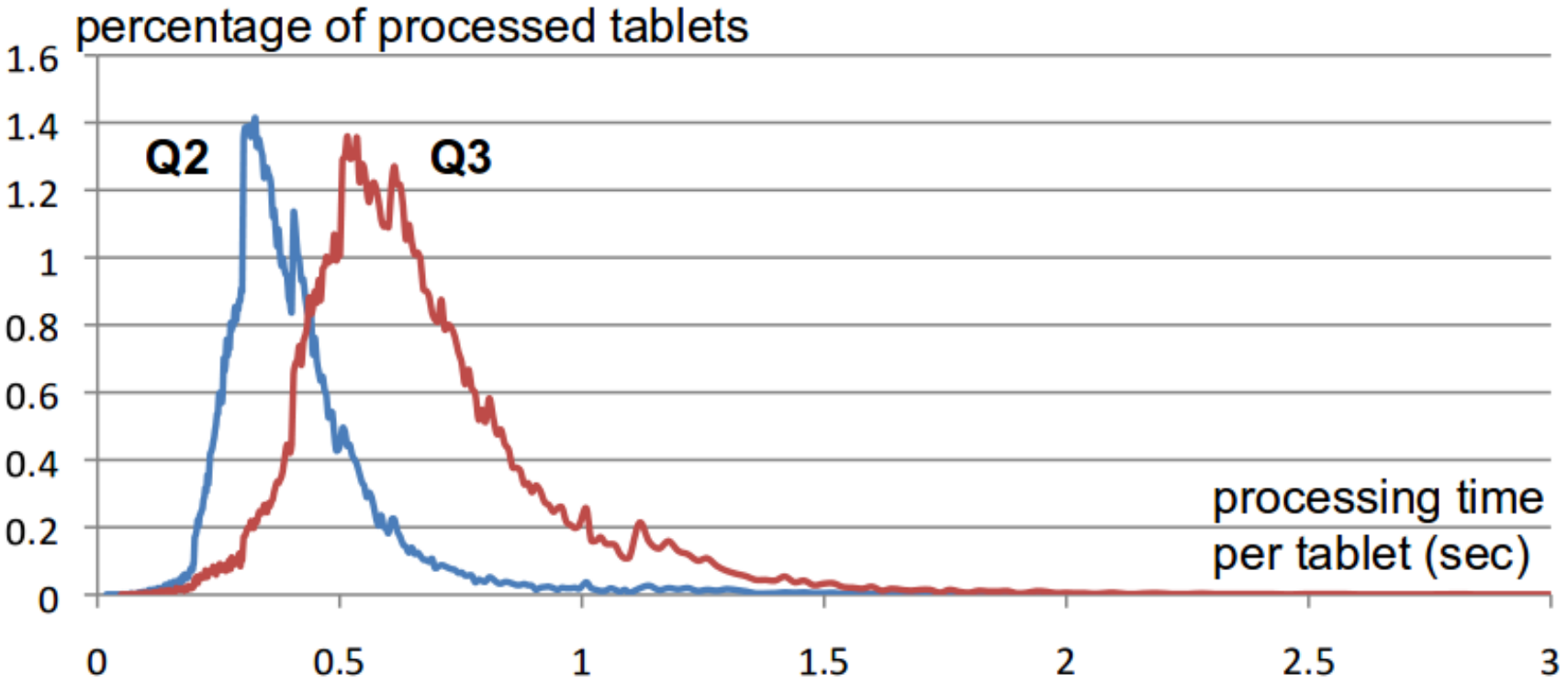
Serving Tree Level Impact



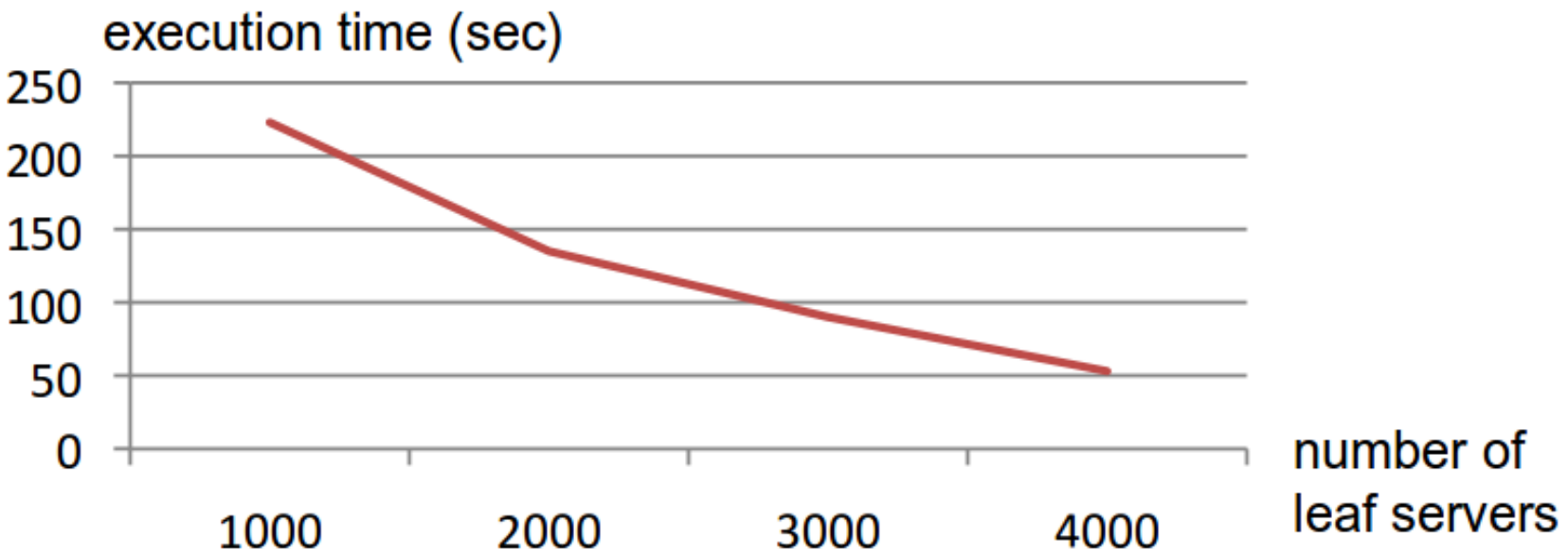
Q_2: SELECT country, SUM(item.amount) FROM T2
GROUP BY country

Q_3: SELECT domain, SUM(item.amount) FROM T2
WHERE domain CONTAINS '.net'
GROUP BY domain

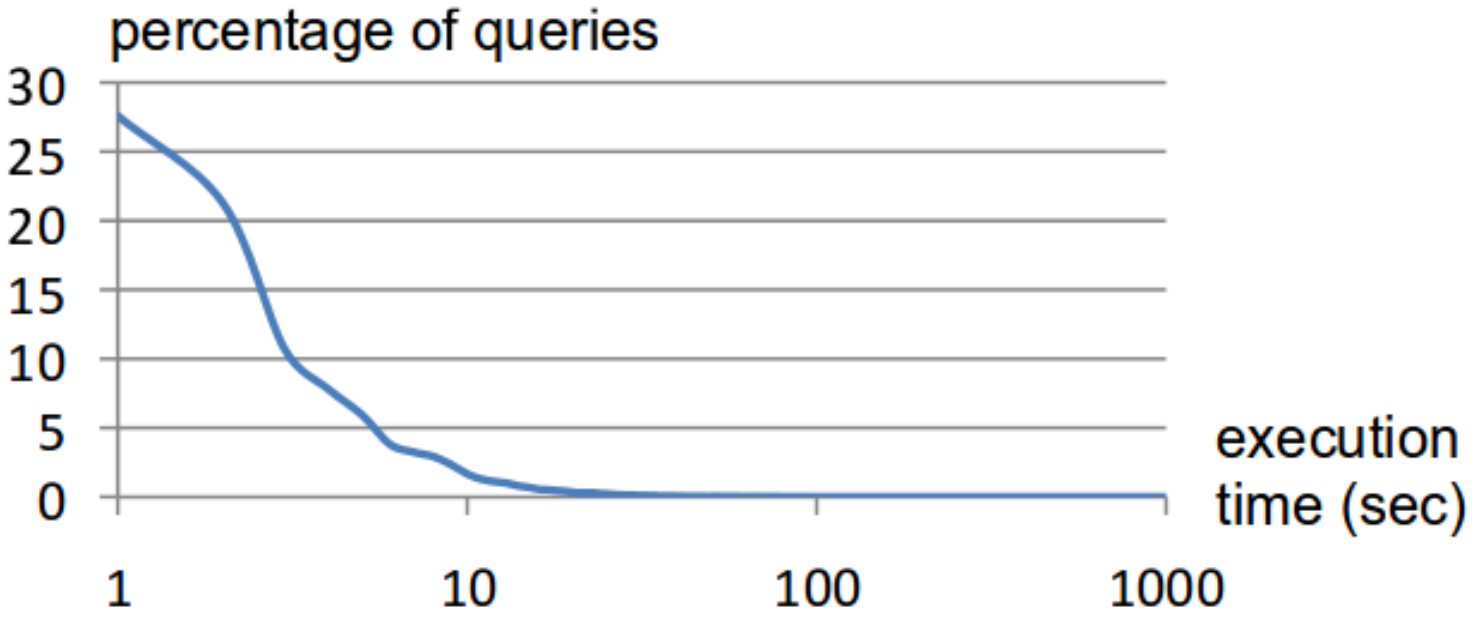
Execution Time Histogram



Scaling Dremel



Query Response Distribution (1 month)



Observations

- Scan based queries can be executed at interactive speeds on disk resident datasets of up to 1 trillion records
- Near linear scalability in the number of columns and servers is achievable for systems containing thousands of nodes
- MR benefits from columnar storage
- Record assembly and parsing are expensive
  - Software layers need to be optimized to directly consume column-oriented database
- In a multi user environment a larger system can benefit from economies of scale while offering a better user experience
- Can terminate queries much earlier and return most of the data to tradeoff speed and accuracy
- Getting to the last few percent within tight time bounds is hard

Related Work

- Large Scale Computing
  - Map Reduce, Hadoop
- Hybrid database/ computation
  - HadoopDB
- Columnar Representation of Nested Data
  - Xmill
- Data Model
  - Complex value models
  - Nested relational models
- Query Language
  - Recursive Algebra and Query Optimizations for Nested Relations
  - Pig
- Parallel Data Processing
  - Scope
  - DryadLINQ

Discussion Topics

- Assumes read-only queries; could this be extended to data cleaning systems that we have seen perviously?
  - Replica consistency issues, etc.
- Protocol buffers was changed to not support optional / required fields. Why might that be?
- How common are queries with “small“ results sets?

Thanks for watching!