Dremel: Interactive Analysis of Web-Scale Database

Presented by Jian Fang

Most parts of these slides are stolen from here: http://bit.ly/HIPzeG

What is Dremel

- Trillion-record, multi-terabyte datasets at interactive speed
  - Scales to thousands of nodes
  - Fault and straggler 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 management tools
  - In situ data access (e.g., GFS, Bigtable)
  - MapReduce pipelines

Widely used inside Google

- Analysis of crawled web documents
- Tracking install data for applications on Android Market
- Spam analysis
- Results of tests run on Google's distributed build system
- Disk I/O statistics for hundreds of thousands of disks
- ……

Outline

- Nested columnar storage
- Query processing
- Experiments
- Observations

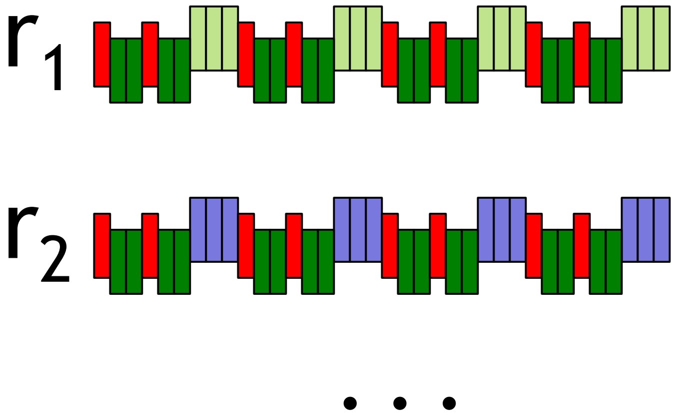
Common Storage Layer

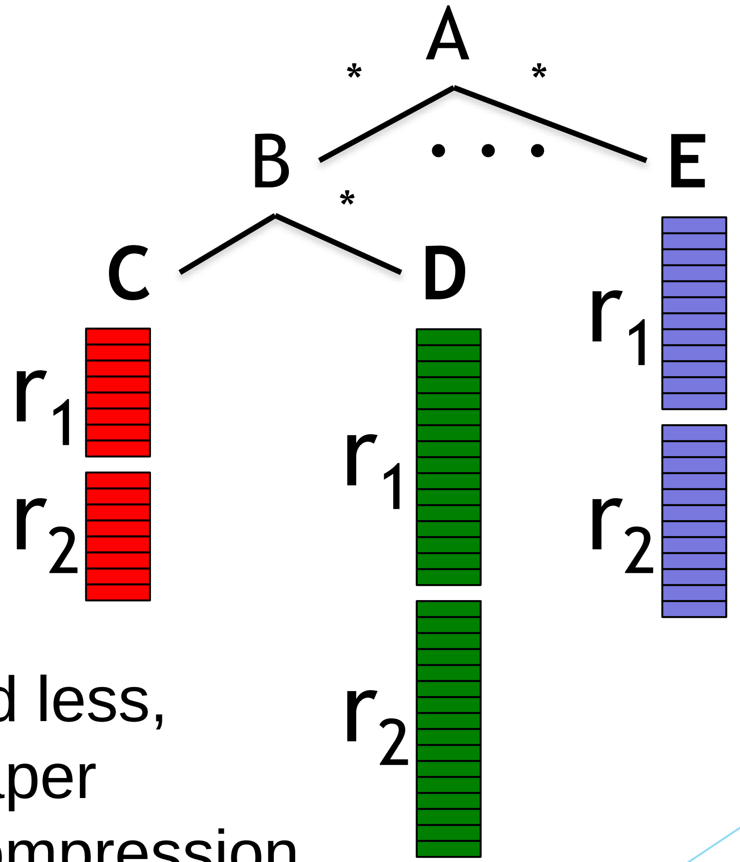
- Google File System
- Fault tolerance
- Fast response time
- Data can be manipulated easily

Rows vs Columns

```
DocId: 10
Links
  Forward: 20
Name
  Language
    Code: 'en-us'
    Country: 'us'
  Url: 'http://A'
Name
  Url: 'http://B'
```

r_1





Read less,
cheaper
decompression

Challenge: preserve structure, reconstruct from a subset of fields

Nested Data Model

```
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;
  }
}
```

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'
```

r_2

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

Repetition and Definition Levels

- Values alone do not convey the structure of a record
- Repetition levels
  - It tells us at what repeated field in the field's path the value has repeated
  - Example: r1, Name.Language.Code
  - Repetition level: [0,2,1,1]

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
  Language
    Code: NULL
  Url: 'http://B'
Name
  Language
    Code: 'en-gb'
    Country: 'gb'
```

R = 0

R = 2

R = 1

R = 1

Repetition and Definition Levels

- Definition Levels
  - Specifying how many fields in a path that could be undefined are actually present in the record
  - Example: Name.Language.Country

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'
```

Missing Country,
d = 2

Missing Language,
d = 1

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

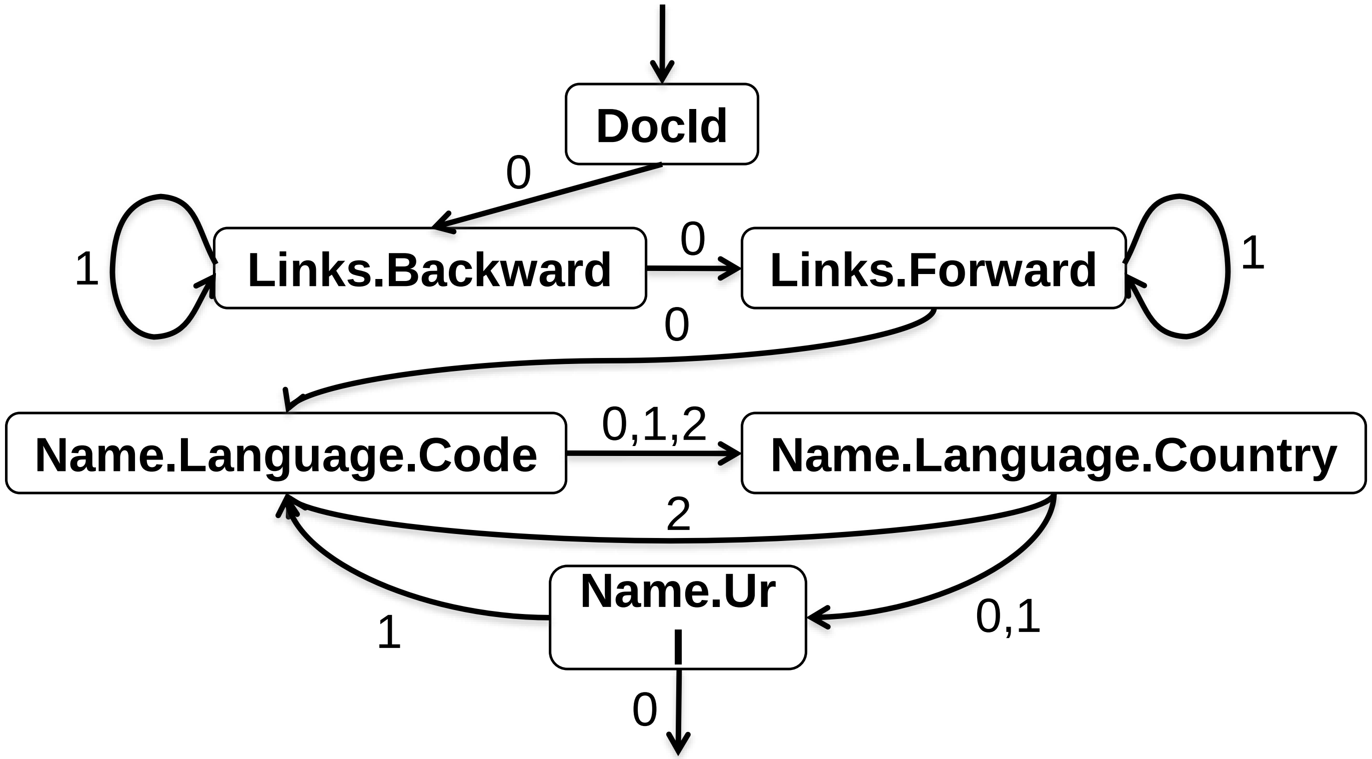
Name.Language.Code

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

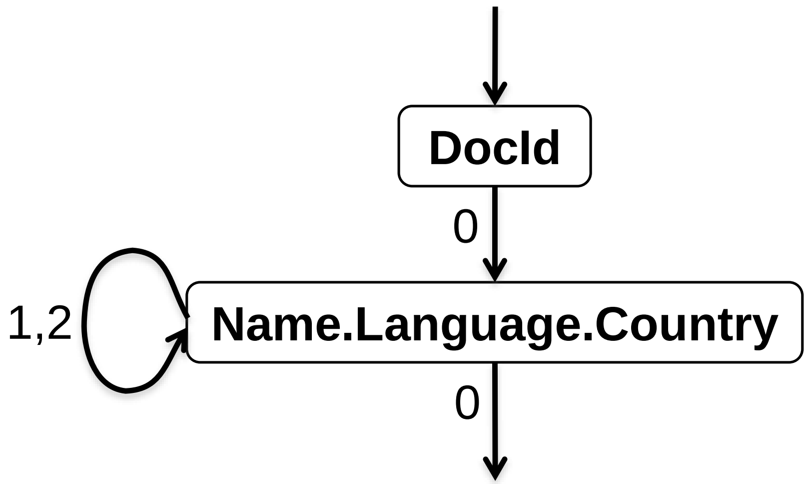
Record Assembly FSM




```
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;
  }
}
```

Transitions labeled with repetition levels

Reading two fields



S_1

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

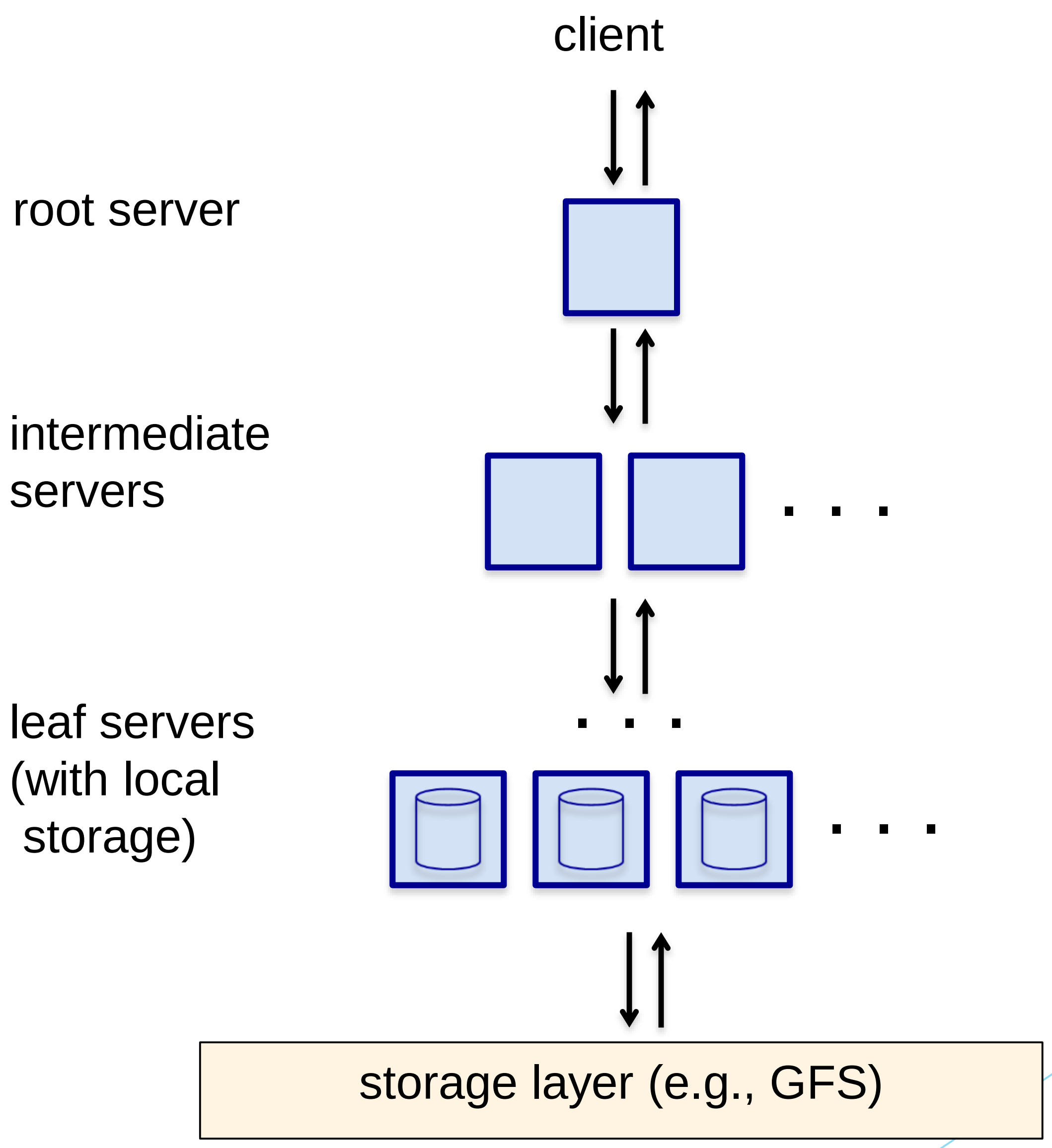
S_2

```
DocId: 20
Name
```

Query Processing

- Optimized for select-project-aggregate
  - Very common class of interactive queries
  - Single scan
  - Within-record and cross-record aggregation
- Approximations: count(distinct), top-k
- Joins, temp tables, UDFs/TVFs, etc.

Serving Tree



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

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

10x speedup
using columnar
storage

2-4x overhead of
using records

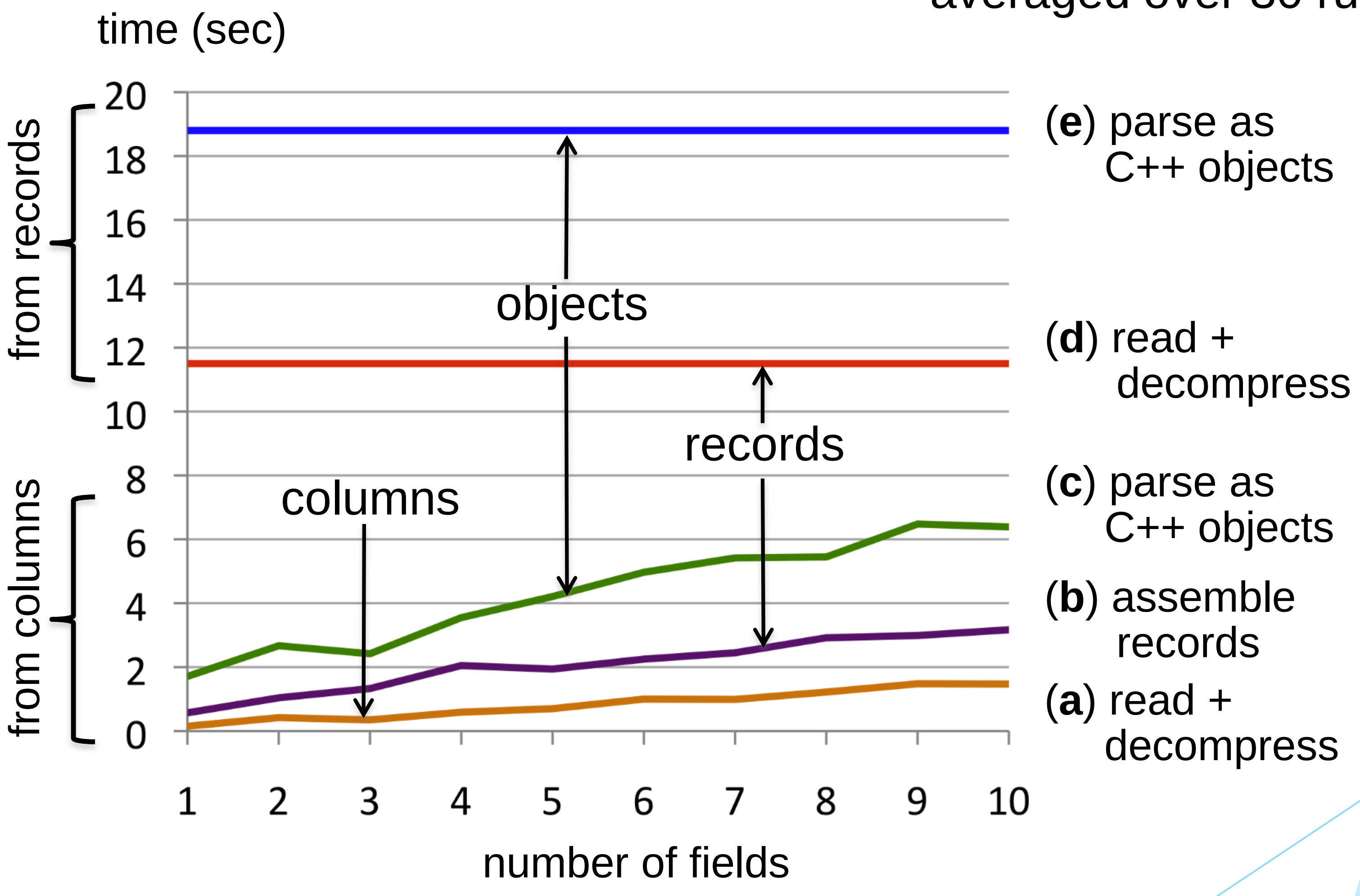
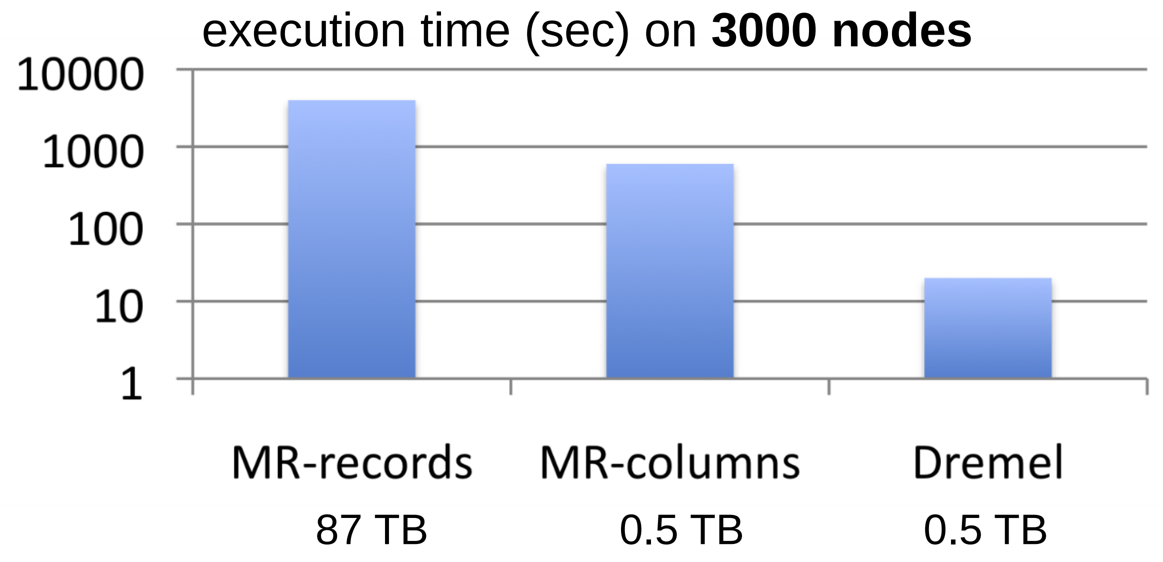


Table partition: 375 MB (compressed), 300K rows, 125 columns

MapReduce and Dremel Execution

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



Sawzall program ran on MR:

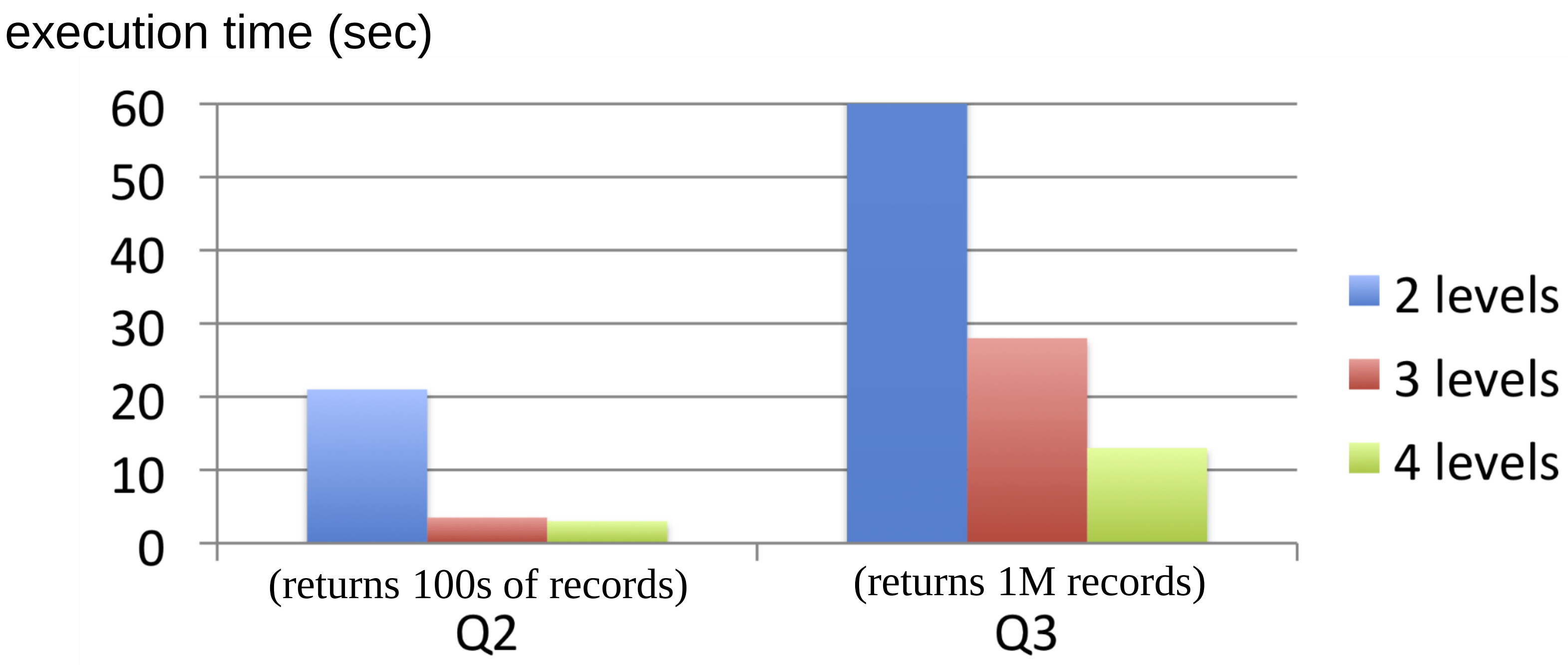
```
num_recs: table sum of int;
num_words: table sum of int;
emit num_recs <- 1;
emit num_words <-
  count_words(input.txtField);
```

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
```

40 billion nested items

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

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

More Information

- Big Query: http://code.google.com/apis/bigquery/
- Apache Drill: http://incubator.apache.org/drill/index.html

Thank You