Dremel: Interactive Analysis of Web-Scale Datasets

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

Google

Presented by:
Sameer Agarwal
sameerag@cs.berkeley.edu

Dremel: Interactive Analysis of Web-Scale Datasets

Interactive Queries on Large Data

- **Input/Output**
  - Sequentially reading a Terabyte from disk in a second requires ~20,000 parallel reads!
- **Processing**
  - CPU-intensive queries may need to run on thousands of cores to complete within a second.
  - Dealing with failures and stragglers is essential.

Interactive Queries on Large Data

- **Input/Output**
  - Sequentially reading a Terabyte from disk in a second requires ~20,000 parallel reads! [Nested Columnar Storage]
- **Processing**
  - CPU-intensive queries may need to run on thousands of cores to complete within a second.
  - Dealing with failures and stragglers is essential.

Interactive Queries on Large Data

- **Input/Output**
  - Sequentially reading a Terabyte from disk in a second requires ~20,000 parallel reads! [**Nested Columnar Storage**]
- **Processing**
  - CPU-intensive queries may need to run on thousands of cores to complete within a second. [**Hierarchical Query Processing**]
  - Dealing with failures and stragglers is essential.

Interactive Queries on Large Data

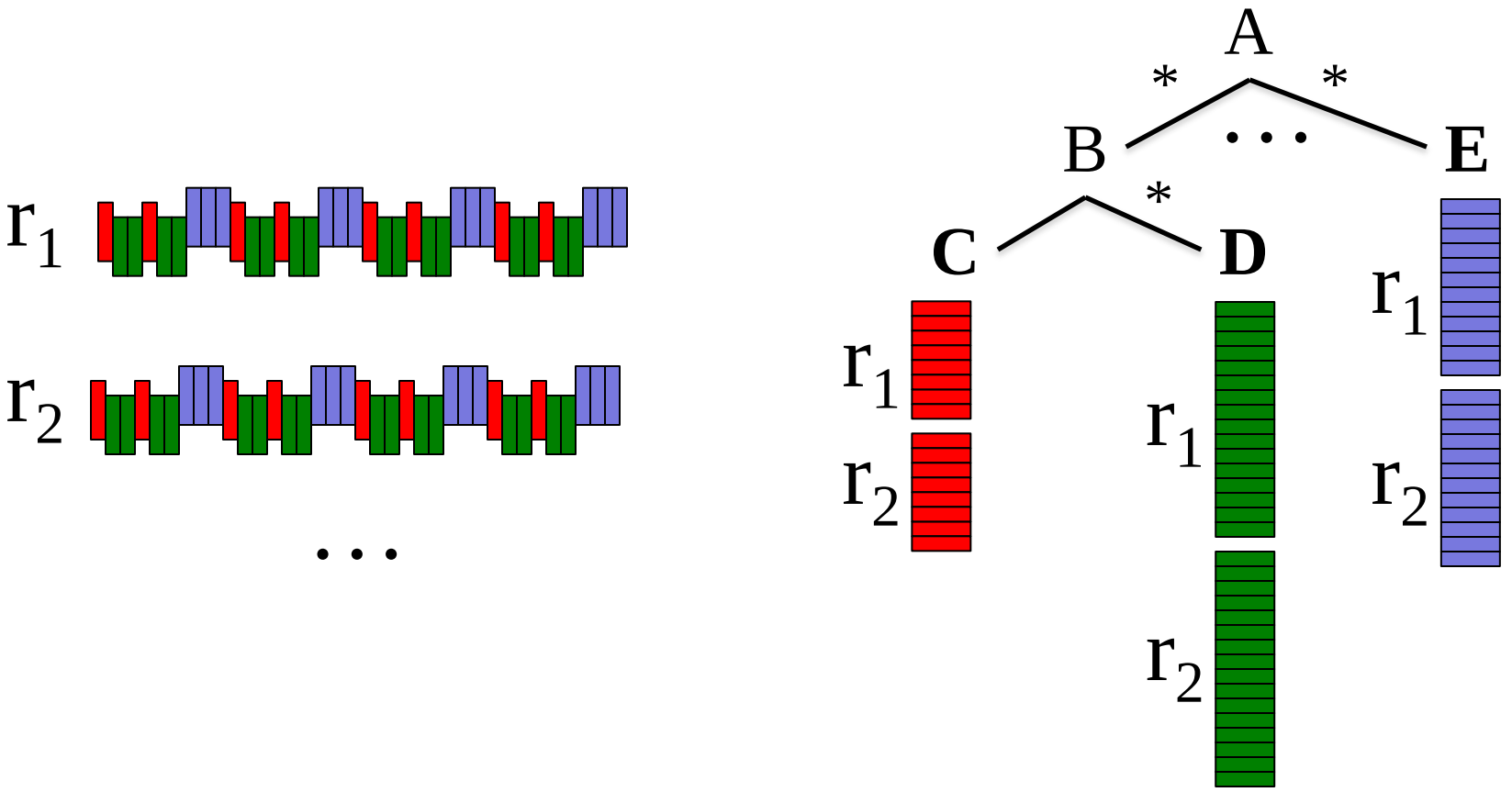
- **Input/Output**
  - Sequentially reading a Terabyte from disk in a second requires ~20,000 parallel reads! [Nested Columnar Storage]
- **Processing**
  - CPU-intensive queries may need to run on thousands of cores to complete within a second. [Hierarchical Query Processing]
  - Dealing with failures and stragglers is essential. [Profiles, Duplicates or Ignores Them]

Nested Columnar Storage

r_1

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

Nested Columnar Storage



Read Less; Cheaper Decompression!

Nested Columnar Storage

```
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: 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'
```

Nested Columnar Storage

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

Links.Forward

value	r	d
20	0	2
40	1	2
60	1	2

```
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'
```

Name.Language.Code

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

Name.Language.Country

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

Building Columns

Name.Language.Code

value	r	d
en-us	0	2

r_1.**Name**$_1$.**Language**$_1$.Code: `'en-us'`

```
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'
```

Repetition (r) and definition (d) levels encode the structural *delta* between the current value and the previous value.

(r): Length of common path prefix
(d): Number of fields in the path that could be optional but are actually present

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

Building Columns

Name.Language.Code

value	r	d
en-us	0	2
en	2	2

r_1.**Name$_1$.Language$_1$**.Code: `'en-us'`

r_1.**Name$_1$.Language$_2$**.Code: `'en'`

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'
```

Building Columns

Name.Language.Code

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

r_1.**Name**$_1$.**Language**$_1$.Code: `'en-us'`

r_1.**Name**$_1$.**Language**$_2$.Code: `'en'`

r_1.**Name**$_2$

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'
```

Building Columns

Name.Language.Code

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

r_1.**Name**$_1$.**Language**$_1$.Code: `'en-us'`

r_1.**Name**$_1$.**Language**$_2$.Code: `'en'`

r_1.**Name**$_2$

r_1.**Name**$_3$.**Language**$_1$.Code: `'en-gb'`

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'
```

Building Columns

Name.Language.Code

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

r_1.**Name**$_1$.**Language**$_1$.Code: `'en-us'`

r_1.**Name**$_1$.**Language**$_2$.Code: `'en'`

r_1.**Name**$_2$

r_1.**Name**$_3$.**Language**$_1$.Code: `'en-gb'`

r_2.**Name**$_1$

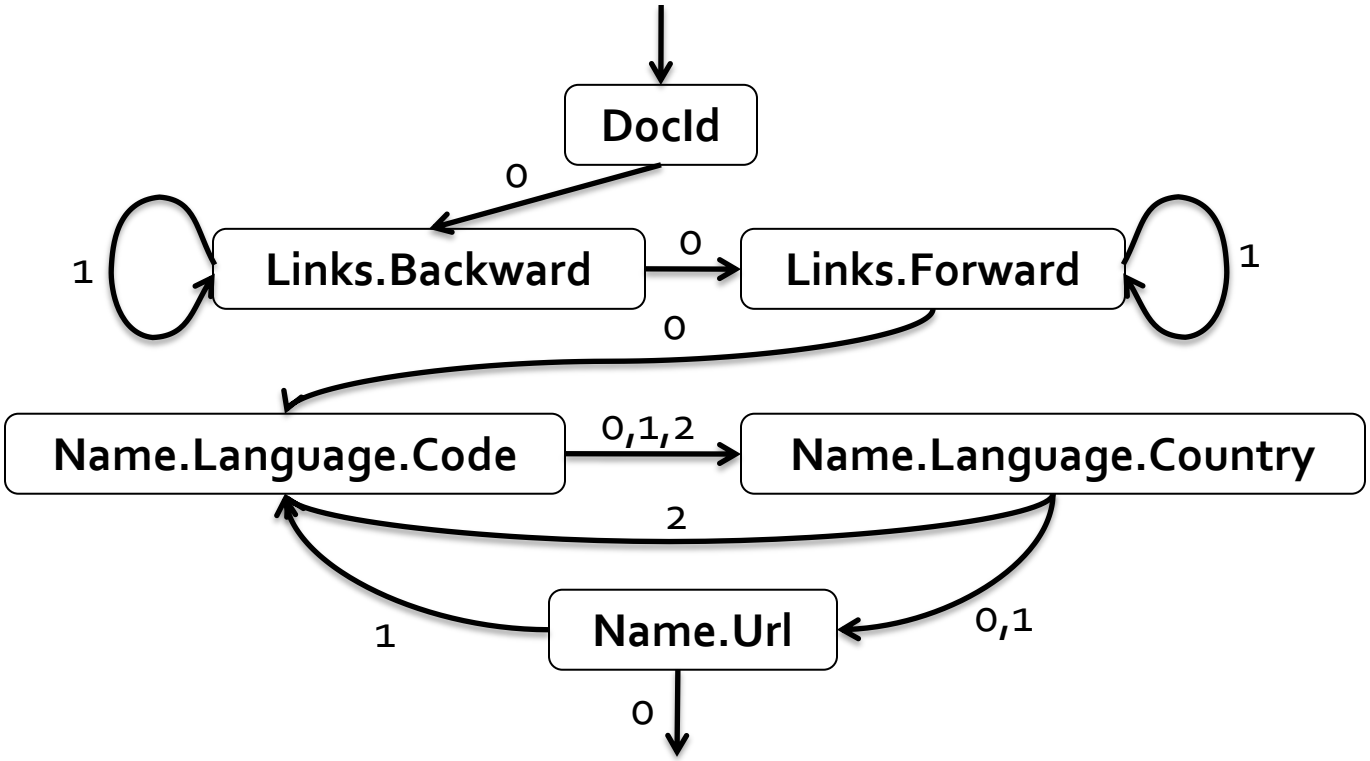
r_1

```
DocId: 10
Links
  Forward: 20
  Forward: 40
  Forward: 60
Name
  Language
    Code: 'en-us'
    Country: 'us'
  Language
    Code: 'en'
  Url: 'http://A'
Name
  Url: 'http://B'
Name
  Language
    Code: 'en-gb'
    Country: 'gb'
```

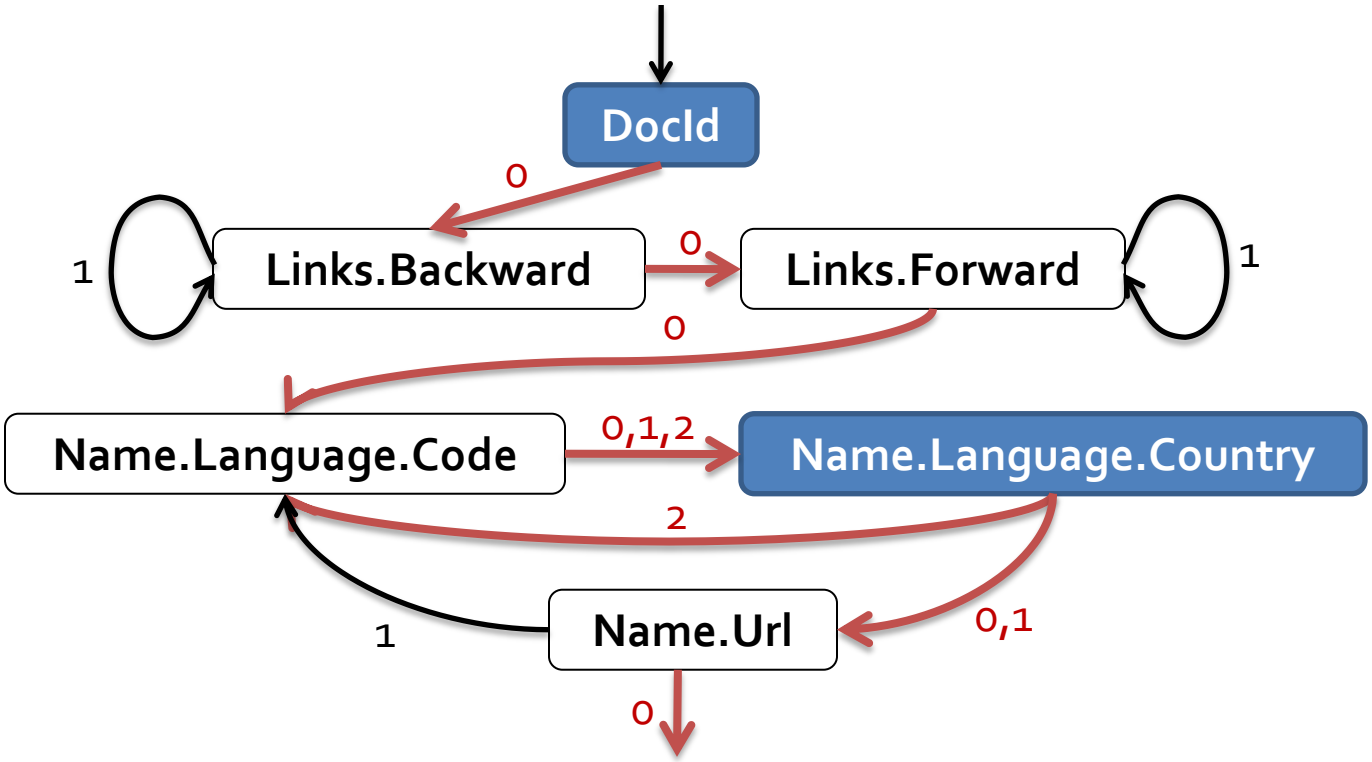
r_2

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

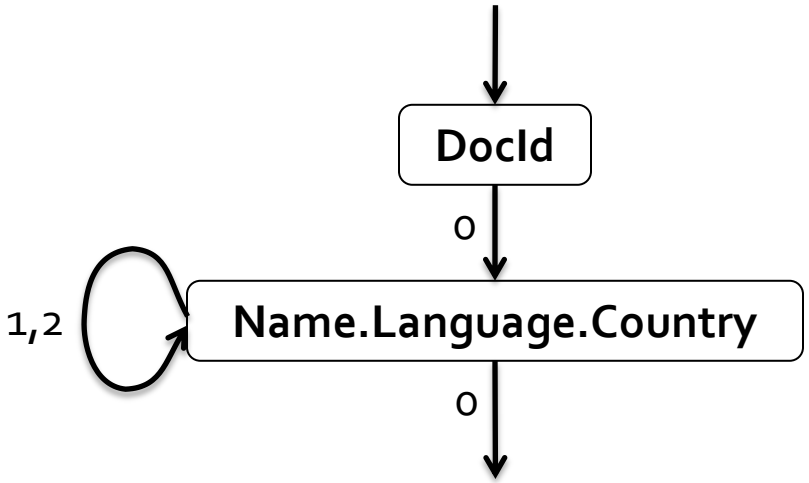
Retrieving Columns



Retrieving Columns



Retrieving Columns



DocId

value	r	d
10	0	0
20	0	0

Name.Language.Country

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

Retrieving Columns

DocId

value	r	d
10	0	0
20	0	0

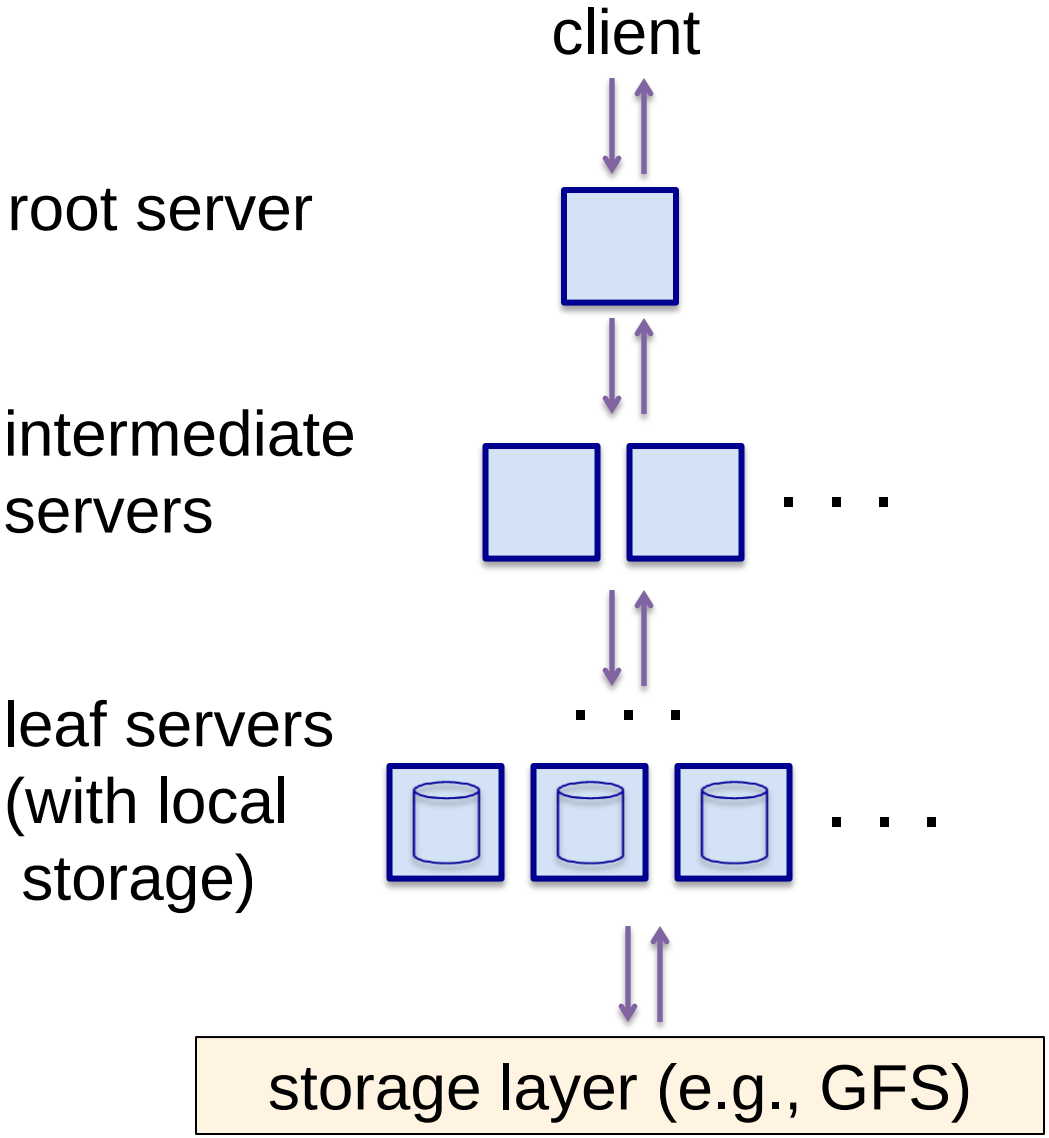
Name.Language.Country

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

```
DocId: 10                    S1
Name
  Language
    Country: 'us'
  Language
Name
Name
  Language
    Country: 'gb'
```

```
DocId: 20                    S2
Name
```

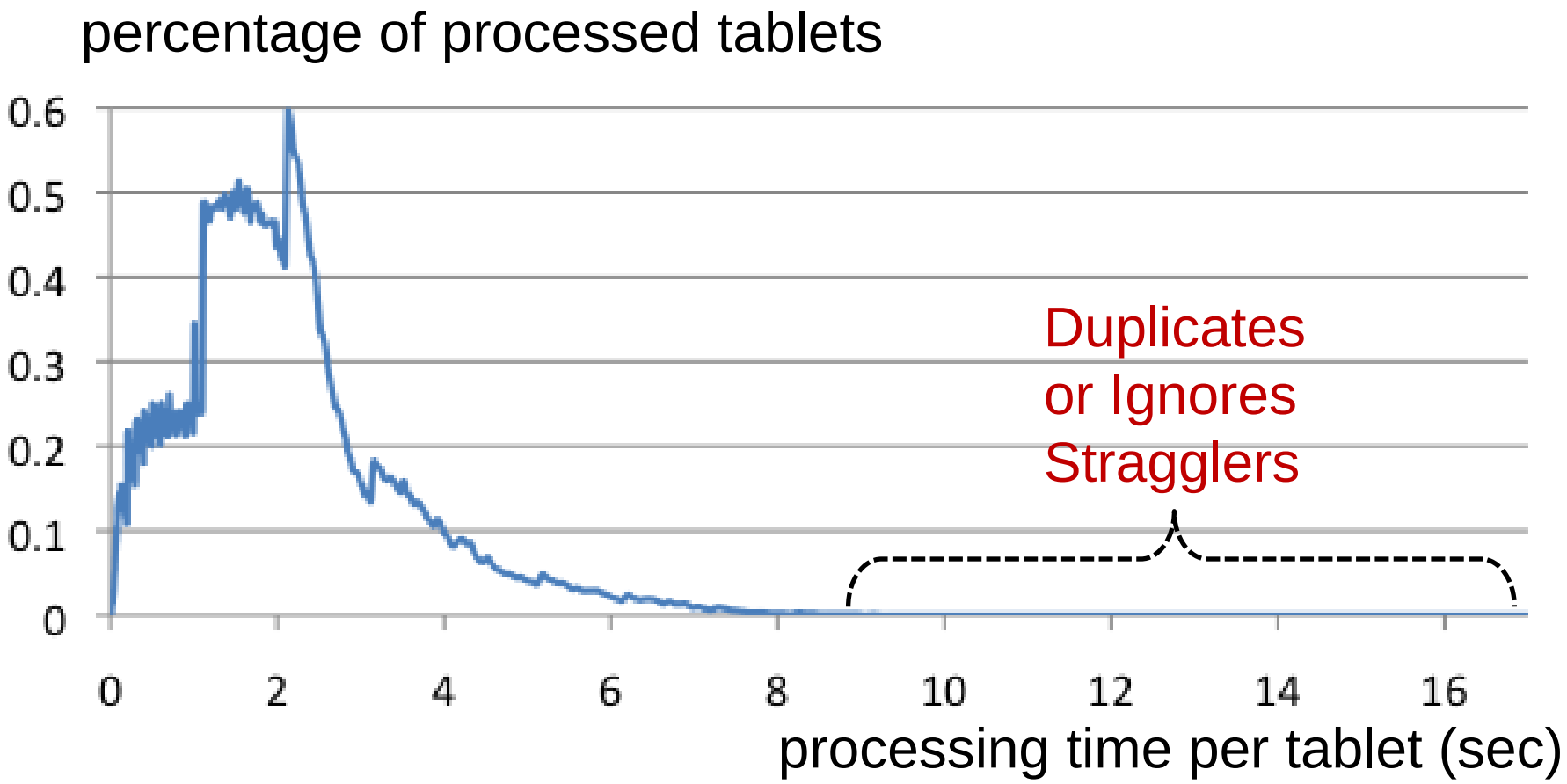
Hierarchical Query Processing



Hierarchical Query Processing

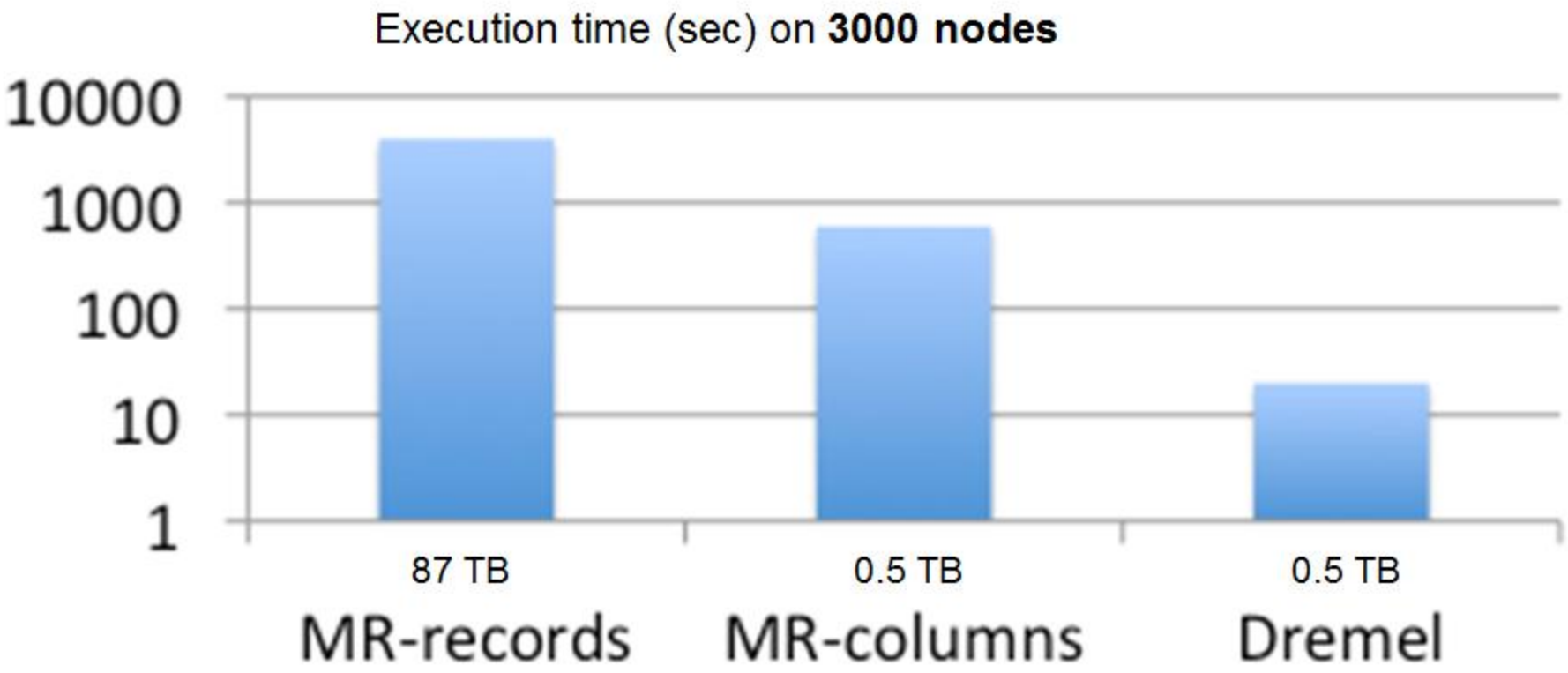
- Optimized for Select-Project-Aggregate queries.
  - Single Scan over Data
  - Recursive Reducers
- Defers discussion of joins, indexing, updates etc. to future work.
- Scheduler's Secret Sauce.

Duplicate/Ignore Stragglers



Comments/Critiques

Does Dremel really require a new execution engine?



What's really novel about Aggregation Trees?

- Very similar to the MapReduce model (Leaf servers run Map tasks and Aggregators are Reduce tasks)
- Partial Aggregates/Recursive Reducers have already been proposed by Traditional Databases as well as SCOPE/Dryad.

Can we make other tradeoffs?

- **Input/Output**
  - Sequentially reading a Terabyte from disk in a second requires ~20,000 parallel reads!
- **Processing**
  - CPU-intensive queries may need to run on thousands of cores to complete within a second.
  - Dealing with failures and stragglers is essential.

Can we make other tradeoffs?

- **Input/Output**
  - Sequentially reading a Terabyte from disk in a second requires ~20,000 parallel reads! [**Sampling? In-memory RDDs?**]
- **Processing**
  - CPU-intensive queries may need to run on thousands of cores to complete within a second.
  - Dealing with failures and stragglers is essential.

Can we make other tradeoffs?

- **Input/Output**
  - Sequentially reading a Terabyte from disk in a second requires ~20,000 parallel reads! [**Sampling? In-memory RDDs?**]
- **Processing**
  - CPU-intensive queries may need to run on thousands of cores to complete within a second. [**Better Data Partitioning?**]
  - Dealing with failures and stragglers is essential.

Can we make other tradeoffs?

- **Input/Output**
  - Sequentially reading a Terabyte from disk in a second requires ~20,000 parallel reads! [**Sampling? In-memory RDDs?**]
- **Processing**
  - CPU-intensive queries may need to run on thousands of cores to complete within a second. [**Better Data Partitioning?**]
  - Dealing with failures and stragglers is essential. [**Giving Answers with Bounded Errors/Confidence Intervals?**]

Thank You!