Dremel: Interactive Analysis of Web-Scale Datasets

CS 744 BIG DATA

PHIL MARTINKUS

Motivation

Large Scale Data must be accessible to analysts and engineers

Interactive queries are important for data exploration, monitoring, online customer support, rapid prototyping, debugging of data, and other tasks

Many databases require a costly loading phase

Web data is often non-relational

The Solution: Dremel

Dremel is a system that supports interactive analysis of very large datasets over shared clusters of commodity machines.

Has been in production at Google since 2006

Can operate on data in place using a distributed storage system

Uses a novel columnar storage format for nested data

Provides a high-level SQL-like language for interactive queries

Data Model

Strongly-typed nested records

Records consist of one or more fields

Fields can be required, optional or repeated

Data Model Example

Each Record represents a document

Required DocId field

Links is an optional group with two nested repeated fields.

Name is a repeated group with a nested Language group.

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'
```

```
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_2

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

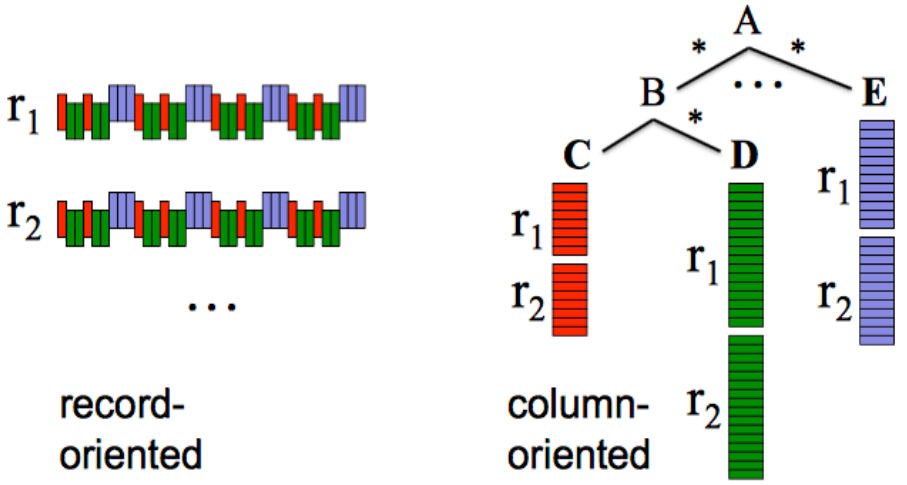
Nested Columnar Storage

All values of a field are stored consecutively in blocks

Goals for the storage system:

- Lossless representation of record structure in columnar format
- Fast encodings
- Efficient record assembly

Repeated records are handled with repetition and definition levels



Repetition Levels

Used to disambiguate occurrences of the same field within the same record

Tell us at what repeated field in the field's path the value has repeated

Repetition Level 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
10	0
20	0

Name.Url

value	r
http://A	0
http://B	1
NULL	1
http://C	0

Links.Forward

value	r
20	0
40	1
60	1
80	0

Links.Backward

value	r
NULL	0
10	0
30	1

Name.Language.Code

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

Name.Language.Country

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

Definition Levels

Whenever an optional or repeated field is not present in a record, the system stores a NULL.

Tell us how many fields in the field's path that could be undefined (because they are optional or repeated) are actually present in the record.

Mostly useful for distinguishing NULL values.

Definition Level 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

Splitting Records into Columns

Recursive algorithm computes levels for each field

A tree of field writers match the structure of the field schema

Many datasets at Google are sparse

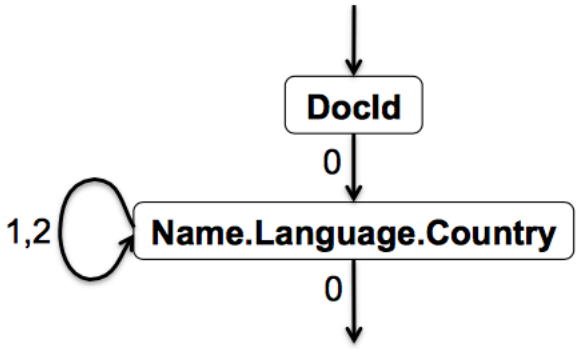
Record Assembly

Goal is to reconstruct records given a subset of fields

Finite state machine (FSM) reads values and appends to output records

An FSM state corresponds to a field reader

The FSM is traversed from the start state to the end state for each record




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

```
DocId: 20                    S2
Name
```

Name.Language.Country

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

Query Language

Based on SQL

Designed for columnar nested storage

```
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;
```

```
Id: 10                                  t1
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

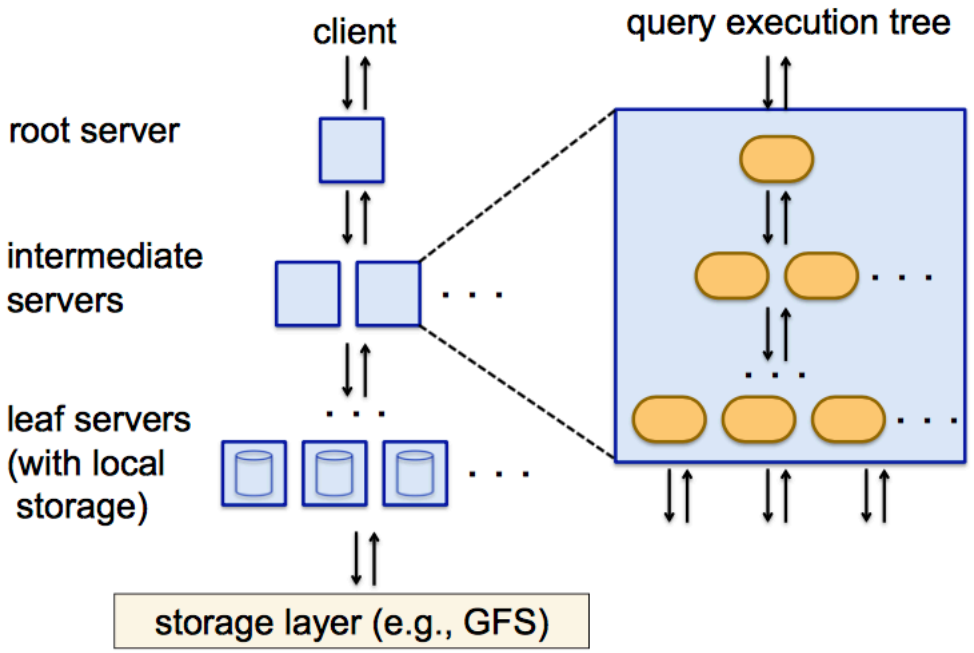
Uses a Tree architecture

Root receives incoming queries

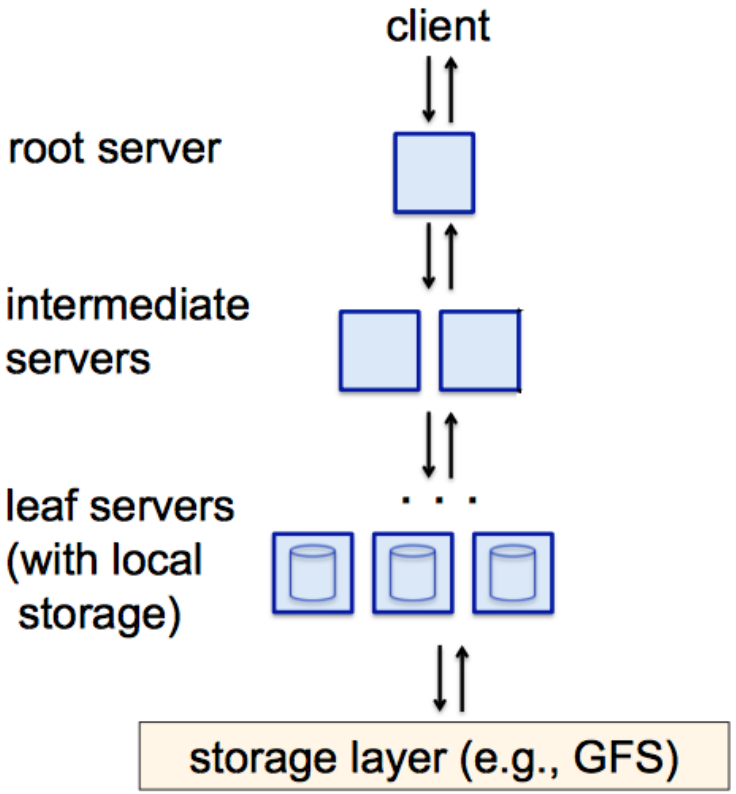
Intermediate servers rewrite the query

Leaf servers access data

Each server has an internal tree corresponding to a physical query execution plan.



Query Execution Example



Query sent to root

SELECT A, COUNT(B) FROM T GROUP BY A

Query is rewritten

SELECT A, SUM(c) FROM (R_1^1 UNION ALL ... R_n^1) GROUP BY A

R_i^1 = SELECT A, COUNT(B) AS c FROM T_i^1 GROUP BY A

Query sent to leaf nodes

A set of iterators scan the input column in lockstep and emit results with annotated repetition and definition levels without actually assembling the records

Query Dispatcher

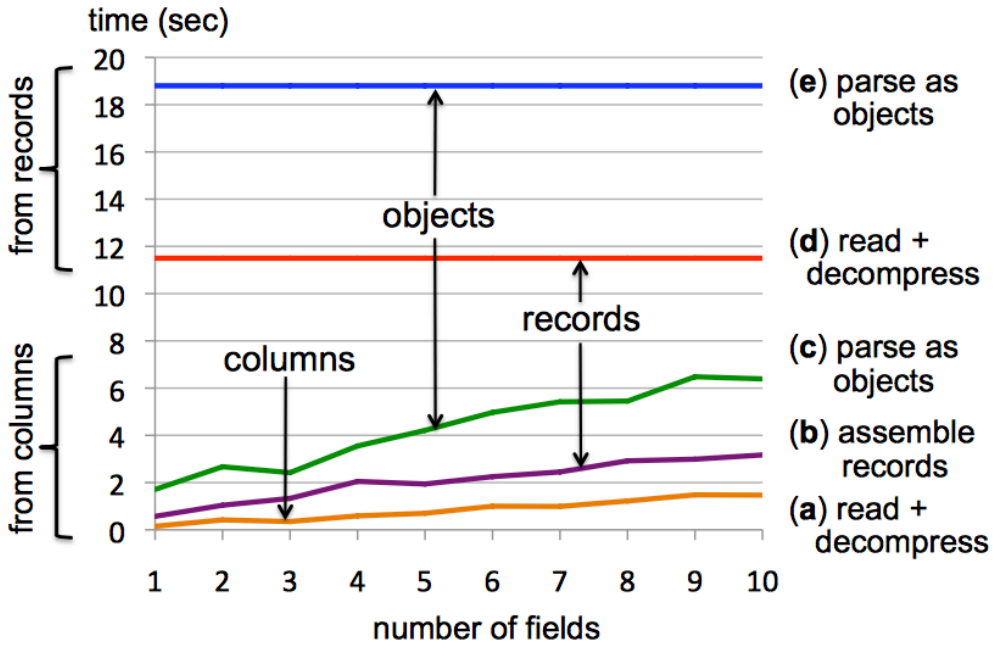
Dremel is a multi-user system that executes queries simultaneously

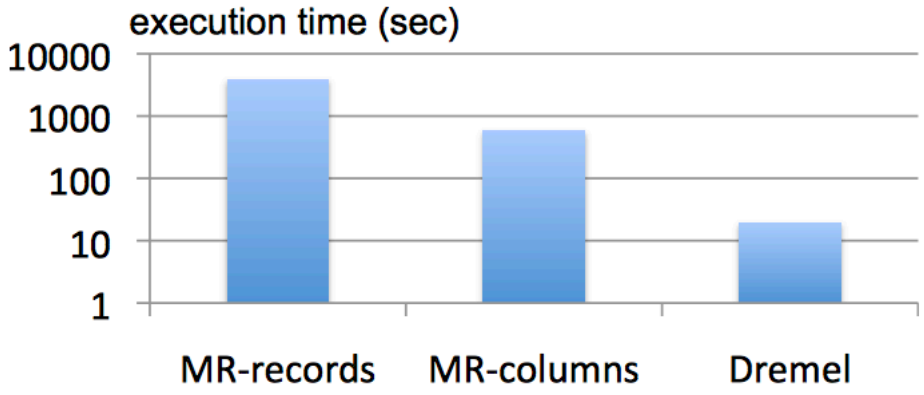
The query dispatcher schedules queries

Dealing with stragglers

- Disproportionally slow processes are rescheduled on another server
- A parameter specifies the minimum percentage of tablets that must be scanned before returning a result

Experiments





Questions?