Dremel

Original slides: Matt Tolton
Modified by: Sarah Chen
Presenter: Sarah Chen
Discussion Leader: Matt Oddo

Large-scale data analysis

Data analysis is lifeblood of many companies

Parallel database systems

- Not designed for extreme scale

MapReduce [Dean, Ghemawat ’04]

- Fault-tolerant data processing

Can be used to execute queries!

- Not designed for low latency (coding, batch jobs)

Dremel

Dremel: data analysis tool, ad-hoc query processing system

Interactive response time required for good data analysis

Key Features

Interactive speed at very large scale

Nested data model with SQL-like language

Interoperates with Google’s data management tools

Applications

Use in complement with MapReduce

1. Run MapReduce to do some data analysis

2. Use Dremel to query resulting output

3. Feed results of query to another MapReduce pipeline or serving system

Widely used 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

10s/1000s-node instances in several data centers

Nested columnar storage (columnio)

Discussion

“The data used in web and scientific computing is non-relational”

This applies to us as researchers. In your own work, what shape does the data come in, and how would you evaluate tradeoffs (e.g. in developer overhead, data loading, etc.) [Michael]

Per group, please share one or two concrete examples of attempts at fitting your data into a relational paradigm.

Nested data model

τ is an atomic data type or record type

Atomic data types are integers, floating-point numbers, strings, etc.

Record types have one or multiple fields

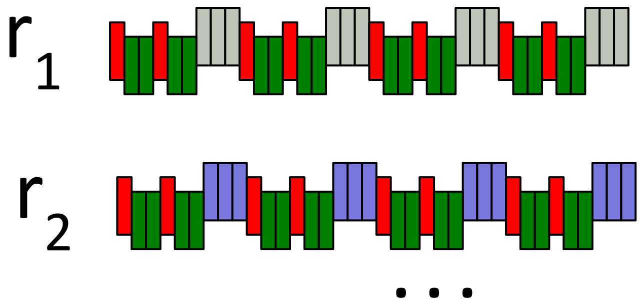
* indicates a repeated field, it can appear more than once

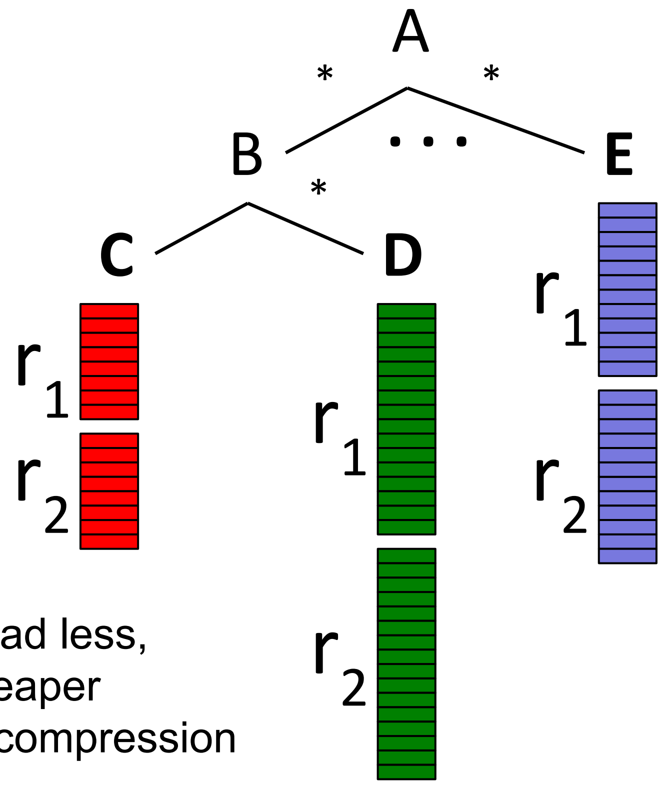
? Indicates an optional field, it can occur 0 or 1 times

Otherwise, the field is required to appear 1 time

$$\tau = \mathbf{dom} \mid \langle A_1 : \tau[*|?], \ldots, A_n : \tau[*|?] \rangle$$

Records vs. columns





Read less,
cheaper
decompression

Challenge: preserve structure, reconstruct from a subset of fields

Nested data model

```
message Document {                            multiplicity:
  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;
  }
}
```

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'
```

ColumnIO 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

Repetition and definition levels

Name.Language.Code

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

$\sqcup$: common prefix

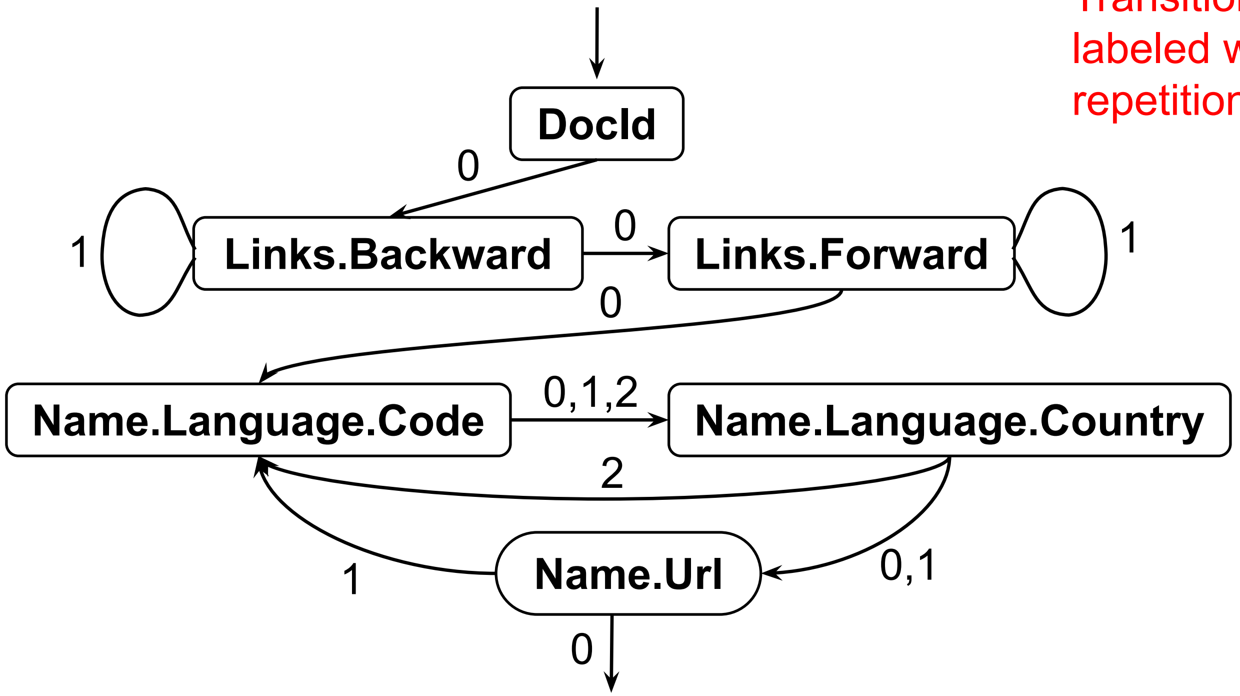
r_1.$Name_1$.$Language_1$.Code: 'en-us'
r_1.$\underbrace{Name_1.Language_2}$.Code: 'en'
r_1.$\underbrace{Name_2}$
r_1.$\underbrace{Name_3.Language_1}$.Code: 'en-gb'
r_2.$Name_1$

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

r: which repeated field has repeated
d: how many fields which could be NULL are present

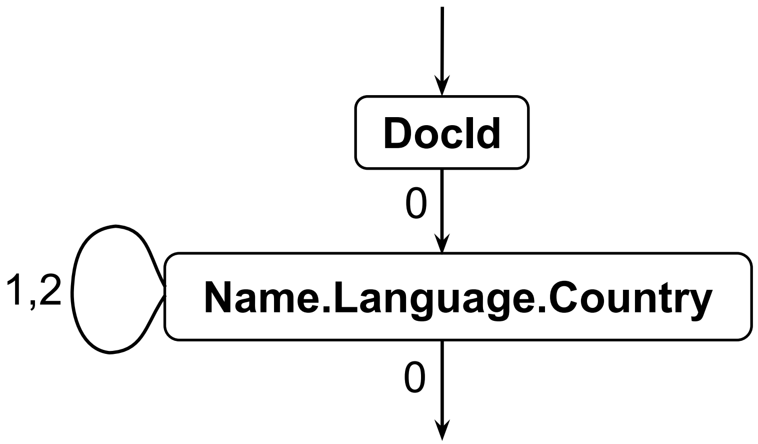
Record assembly FSM



Transitions labeled with repetition levels

For record-oriented data processing (e.g., MapReduce)

Reading two fields




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

```
DocId: 20                    s2
Name
```

Both Dremel and MapReduce can read same columnar data

- Structure of parent fields is preserved.

Hierarchical query processing

Query processing architecture

Optimized for select-project-aggregate

- Very common class of interactive queries
- Single scan
- Within-record and cross-record aggregation

Unit of storage: *tablet*

- Self-contained horizontal partition of a table

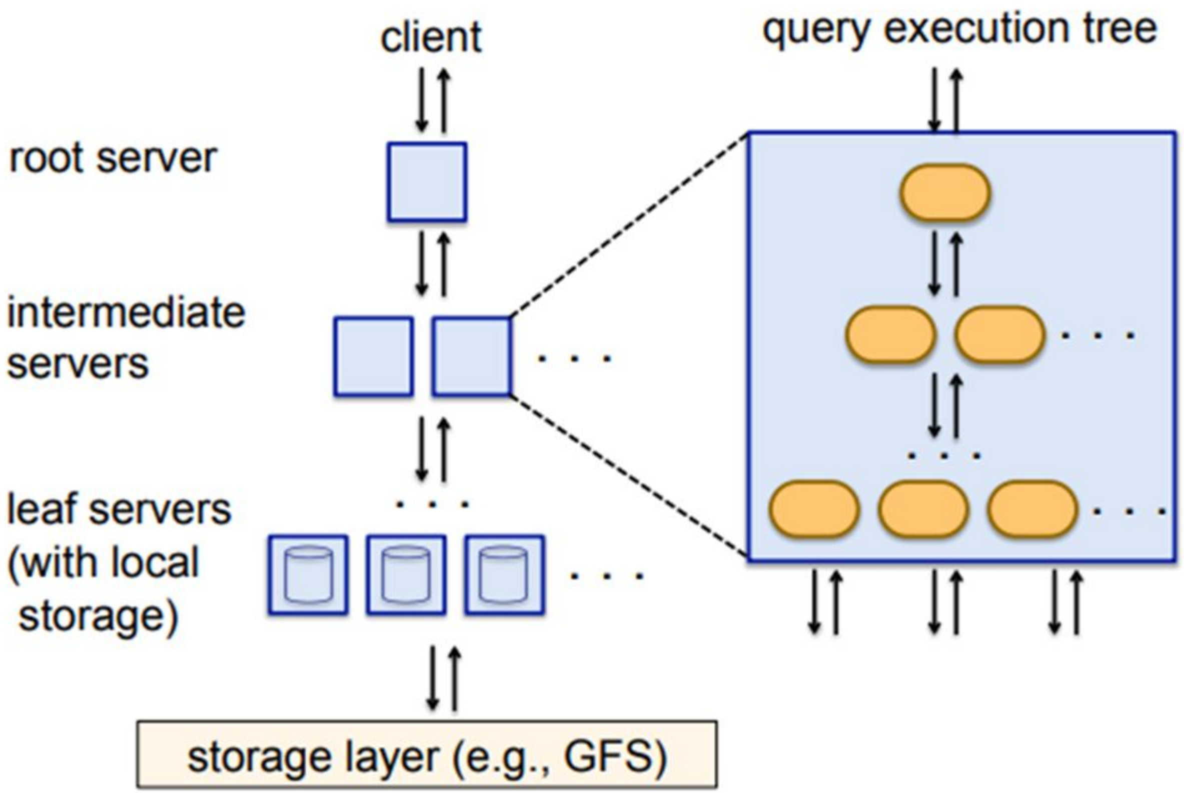
Schema	Metadata	Data		
	keys, order, ranges, …	C_1	…	C_n

Unit of execution: *slot*

- Thread on a server
- E.g., 3K servers × 8 threads = 24K slots

Serving tree

[Dean WSDM'09]



- Parallelizes scheduling and aggregation
- Fault tolerance
- Stragglers
- Designed for "small" results (<1M records)

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

$R_1$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

Dremel: A Decade of Interactive SQL Analysis at Web Scale

Presenter: Sarah Chen
Discussion: Matt Oddo

Dremel

Many key ideas and architectural principles introduced by Dremel have become trends or best practices

1. SQL
2. Disaggregated compute and storage
3. Columnar storage
4. In situ data analysis
5. Serverless computing

Dremel also dealt with latency

Moving Away from SQL- Google

Early 2000s- Big Data era at Google

“SQL doesn’t scale”

Turned to NoSQL systems

Gained scalability, lost ease of use and ability to iterate quickly

Coming back to SQL- Google

Dremel brought SQL back

Faster and simpler to write SQL queries to perform data analysis

Elsewhere at Google, F1 project and other OLTP-focused applications

New challenge- Each system had own dialect

Solution- GoogleSQL project resulting in one SQL dialect, but still issue across industries

Coming back to SQL- Open source world

Followed similar journey

Left SQL due to issues of scalability and cost as data grew

Came back due to challenges of complexity and slow iteration

E.g., HiveSQL, SparkSQL, and Presto

Disaggregation- Storage

Initially, servers with local disks directly attached

Motivation: Scalability

Shifted to Borg and replicated storage organization

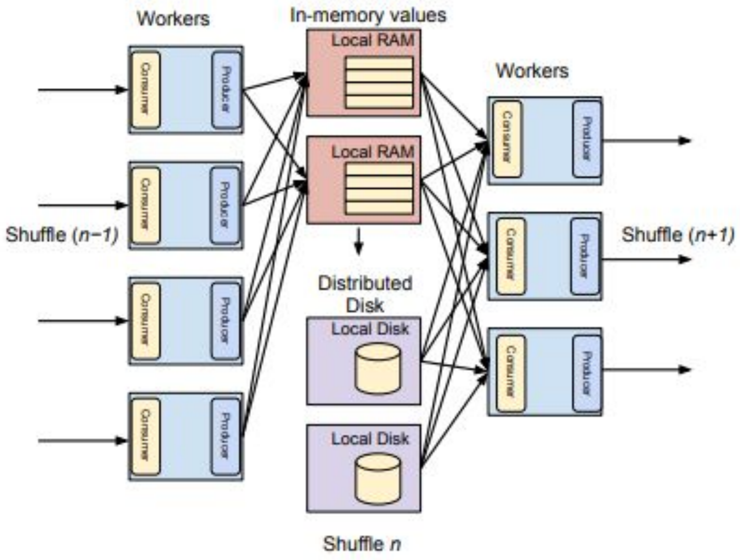
Still had some issues

Solution- Use GFS for storage

Time consuming to get there due to latency, will discuss in more detail later

Disaggregation- Memory

Motivation- Implementing distributed join through shuffle primitive, using local RAM and disks for intermediate storage but could not scale
Result- Shuffle implementation where RAM and storage managed separately
Allowed for in-memory query execution
Big influence on architecture



Disaggregation

Major trend

Can provision resources independently from one another

Better cost-performance and elasticity

Columnar storage for nested data

Early 2000s- Many semi-structured data with flexible schemas as opposed to relational schema

2000s-2010s- Column based storage one of other trends in DBMS research, e.g., Vertica

Dremel proposed columnar storage for semistructured data

Other companies influenced by this such as Twitter and Cloudera, Facebook and Hortonworks, and the Apache Foundation

Columnar storage for nested data

```
DocId: 10                 r1
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;
  repeated group Name {
    repeated group Language {
      required string Code;
      optional string Country;
    }
    optional string Url;
  }
}
```

```
DocId: 20                 r2
Name
  Url: 'http://C'
```

Dremel paper

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

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

ORC proposed by Facebook and Hortonworks

DocId

value	p
10	true
20	true

Name

len
3
1

Name.Url

value	p
http://A	true
http://B	true
	false
http://C	true

Name.Language

len
2
0
1
0

Name.Language.Code

value	p
en-us	true
en	true
en-gb	true

Name.Language.Country

value	p
us	true
	false
gb	true

Columnar storage for nested data

New columnar format- Capacitor

Extensions

1. Efficient filtering
2. Can reorder rows
3. Support for more complex schemas

Original, no RLE runs

State	Quarter	Item
WA	Q2	Bread
OR	Q1	Eggs
WA	Q2	Milk
OR	Q1	Bread
CA	Q2	Eggs
WA	Q1	Bread
CA	Q2	Milk

Reordered

State	Quarter	Item
OR	Q1	Eggs
OR	Q1	Bread
WA	Q1	Bread
WA	Q2	Bread
WA	Q2	Milk
CA	Q2	Milk
CA	Q2	Eggs

Reordered, RLE encoded

State	Quarter	Item
2, OR	3, Q1	1, Eggs
3, WA	4, Q2	3, Bread
2, CA		2, Milk
		1, Eggs

```
message Node {
  optional Payload value;
  repeated Node nodes;
}
```

Discussion

“Table row-store is a legacy paradigm, the future is columnar-store!”

Agree? Disagree?

In situ data analysis

Access data in place without data loading and transformation

Initially, like other DBMSs, Dremel stored data in proprietary format inaccessible to other tools

When transferred to GFS, switched to “open-sourced” self-describing columnar format

Allows other tools and SQL queries to operate on data

In situ data analysis

Built on in two ways

1. Added different file formats
2. Federation, can do in situ analysis with other file systems

In situ data analysis

Drawbacks

- Have to manage data yourself

Need for both in situ data analysis but also managed storage systems

Serverless Computing

Serverless computing- Elastic, multi-tenant, on-demand service

Three key ideas that enabled serverless computing

1. Disaggregation
2. Fault tolerance and responsibility
   - Designed with idea that compute resources unreliable so workers are unreliable as well
   - Enabled easy adjusting of resources
3. Virtual scheduling units

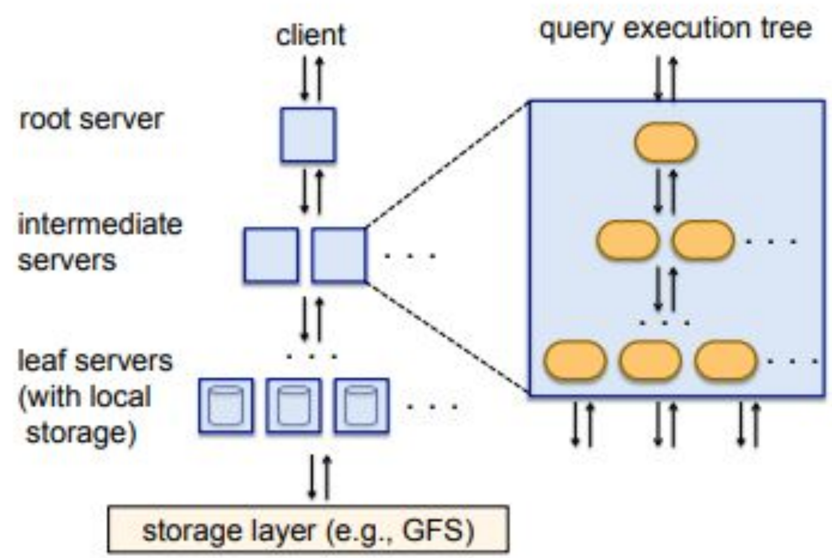
Evolution of Serverless Computing

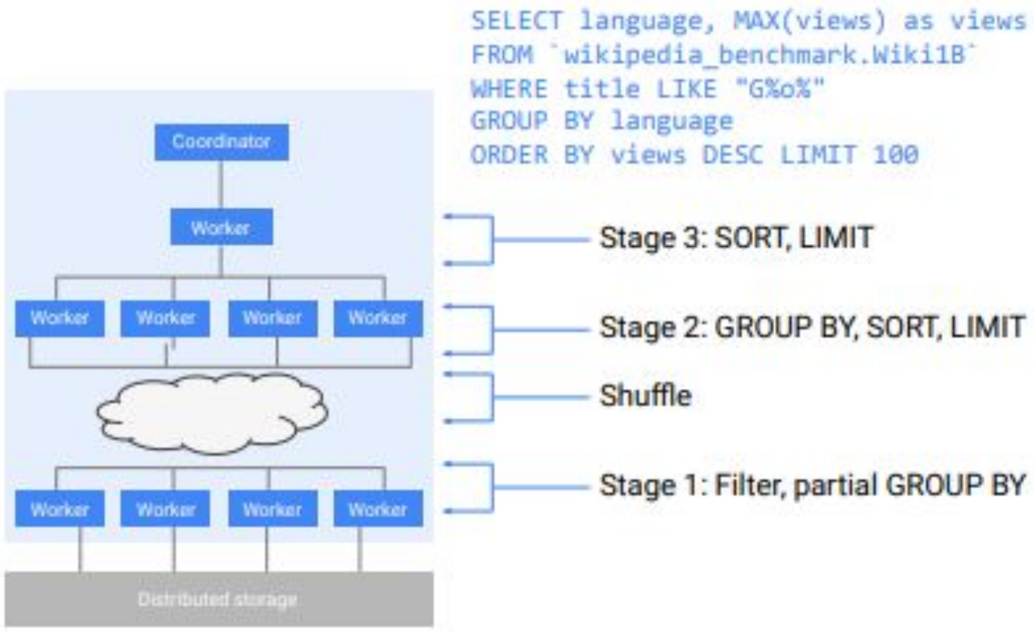
1. Centralized scheduling
   - Previously, query dispatcher for each server node
   - Now, scheduler that uses entire cluster state
2. Shuffle persistence layer
   - Stores results of shuffle
   - Scheduler can adjust number of workers based on result

Evolution of Serverless Computing

3. Flexible Execution DAGs

- Query coordinator first receives query
- Workers are pool without predefined structure





Evolution of Serverless Computing

4. Dynamic Query Execution
   - As execute query plan, can update query execution tree based on statistics
   - Ability to do this due to shuffle persistence layer and centralized query coordinator

Latency

Need low latency for Dremel but some design principles work against latency

Dremel uses many techniques to handle this

- Stand-by server pool
- Speculative execution
- Multi-level execution tree
- Column-oriented schema representation
- Balancing CPU and I/O with lightweight compression
- Approximate results
- Query latency tiers
- Reuse of file operations
- Guaranteed capacity
- Adaptive query scaling

Conclusion

Overall, Dremel got many things right

- Disaggregated compute and storage
- Serverless computing
- Columnar storage for semi-structured data
- In situ data analysis

Also a few things missed

- Shuffle layer
- Managed data option required in addition to in situ
- SQL standards

Discussion

What kind of provisions can be made in the case that instead of the scale of data changes, it is the structure of the data itself that changes? [Nalin]