Dremel

Presenter: Daniel Long
Discussion: Suzuran Takikawa

Slides adapted from:
Matt Tolton, Sarah Chen, Matt Oddo

Introduction - Papers

Papers by: Sergey Melnik

Dremel: Interactive Analysis of Web-Scale Datasets

- VLDB 2010, Describes another “New SQL” approach.

Dremel: A Decade of Interactive SQL Analysis at Web Scale

- VLDB 2020, Test of time award paper.

Dremel: Interactive Analysis of Web-Scale Datasets

Large Scale Data Analysis

Enabled by low cost storage allows for vast quantity of data.

Data analysis is the lifeblood of many companies!

Requires parallelism and flexibility.

Some existing approaches:

- Parallel Databases
- Custom Binaries and Programs
- MapReduce

Dremel

New data analysis Tool: Dremel

- Support interactive analysis of large datasets over cluster of commodity machines.
- Allow “in situ” access of nested data.
- Interoperates with Google’s data processing tools.

Widely used in Google - In Production Since 2006

- 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.

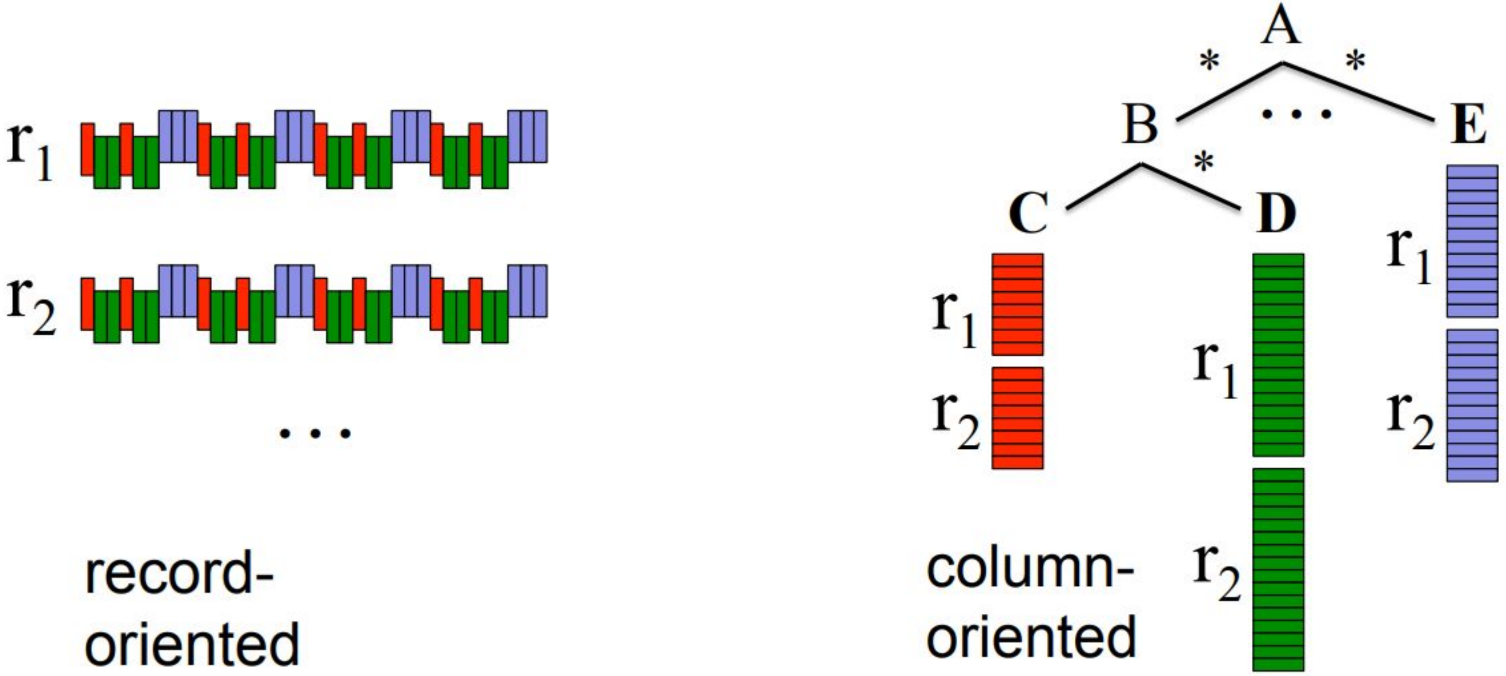
Dremel

Dremel builds on ideas from web search and parallel DBMS.

- Architecture borrows concepts of serving tree.
- Provides high level SQL-like language to express ad hoc queries.
- Column-Striped/Columnar Storage Representation

Columnar Storage Format

Record vs. Column Oriented



Nested Data Model

τ is an atomic type or a record type

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

Records consist of one or multiple fields.

Repeated fields ($*$) may occur multiple times in a record (interpreted as lists of values).

Optional fields (?) may be missing from the record. Otherwise, a field is required, i.e., must appear exactly once.

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

Nested Data Model Example

```
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'
```

Column-stripped Representation

r - Repetition Levels
d - Definition Levels

```
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'
```

```
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

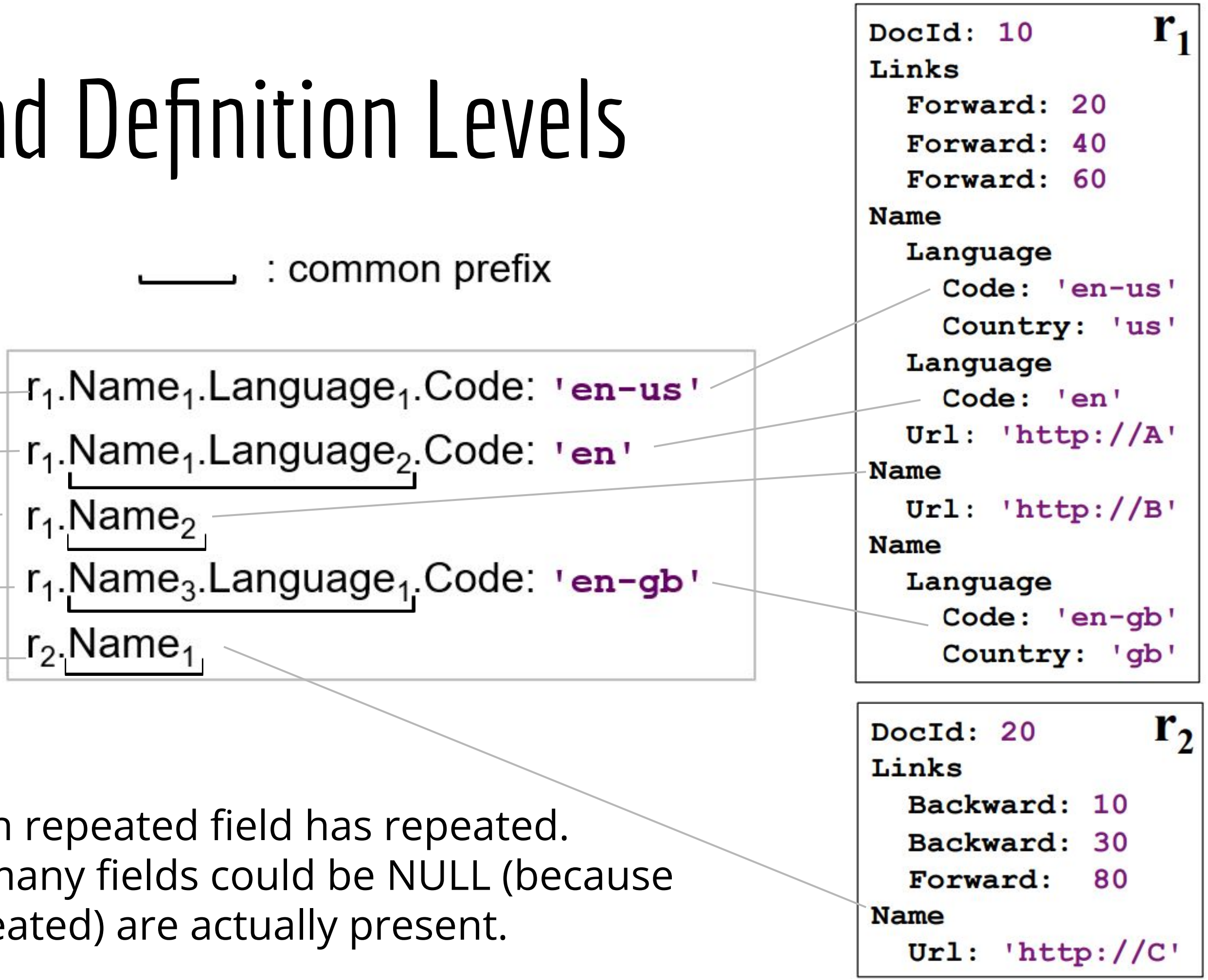
Name.Language.Country

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

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



Repetition levels: Which repeated field has repeated.
Definition levels: how many fields could be NULL (because they are optional or repeated) are actually present.

Fast Encoding

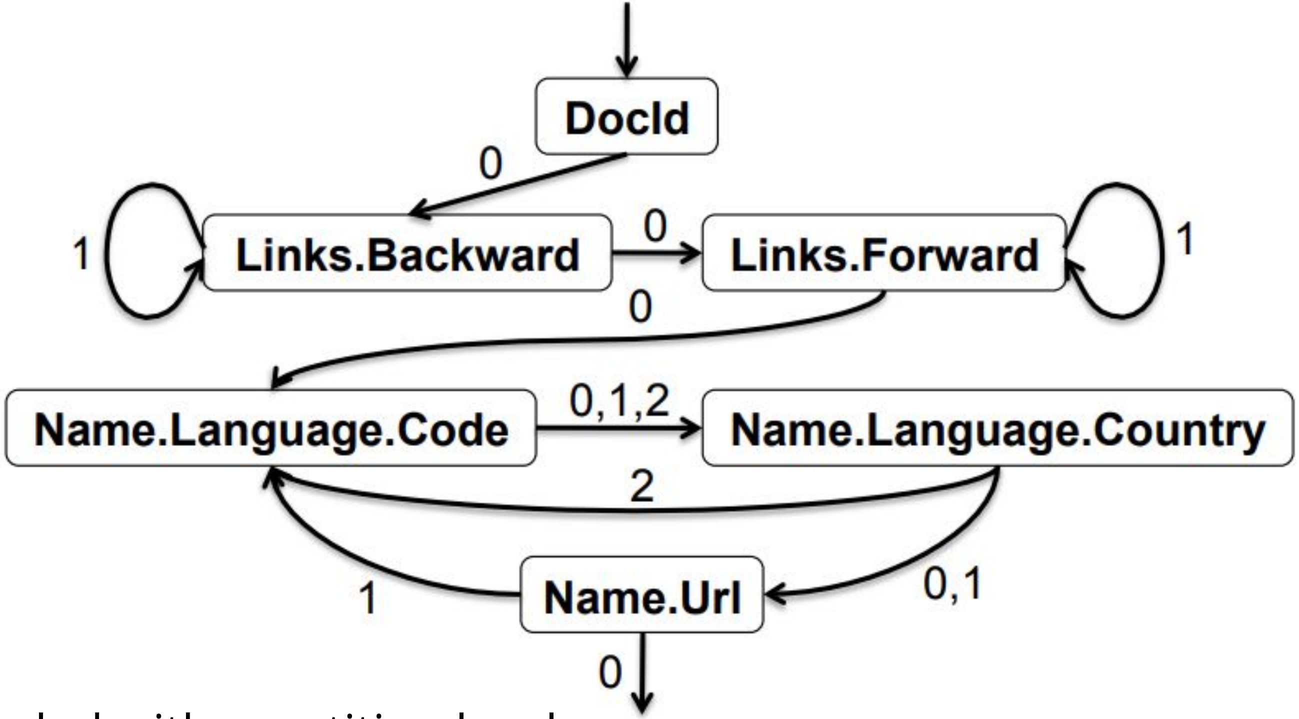
Another aspects of Retrieval Efficiency - Fast Encoding

Create a tree of field writers

- Update field writers only when they have their own data
- Not propagating parent state unless necessary

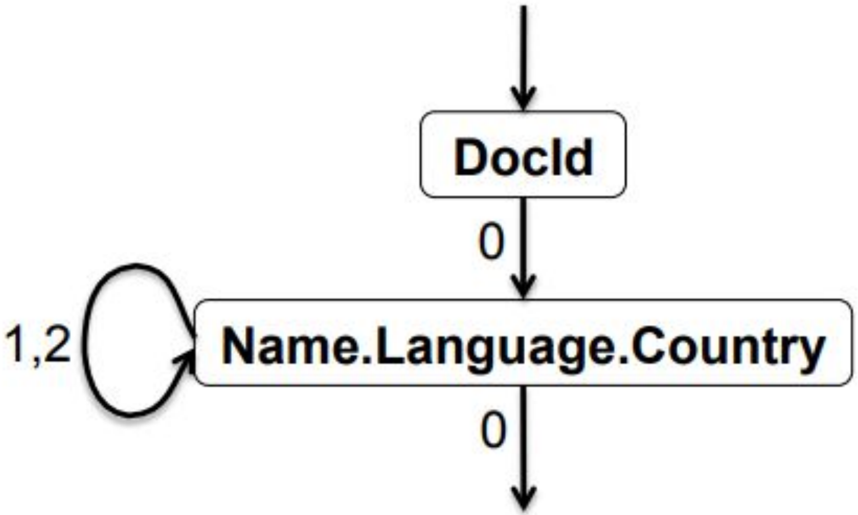
Record Assembly

Needed for record-oriented data processing (Such as MapReduce)



Edges are labeled with repetition levels.

Assembling Records of Two Fields



s_1

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

s_2

```
DocId: 20
Name
```

Sample Dremel Query

```
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; }}}
```

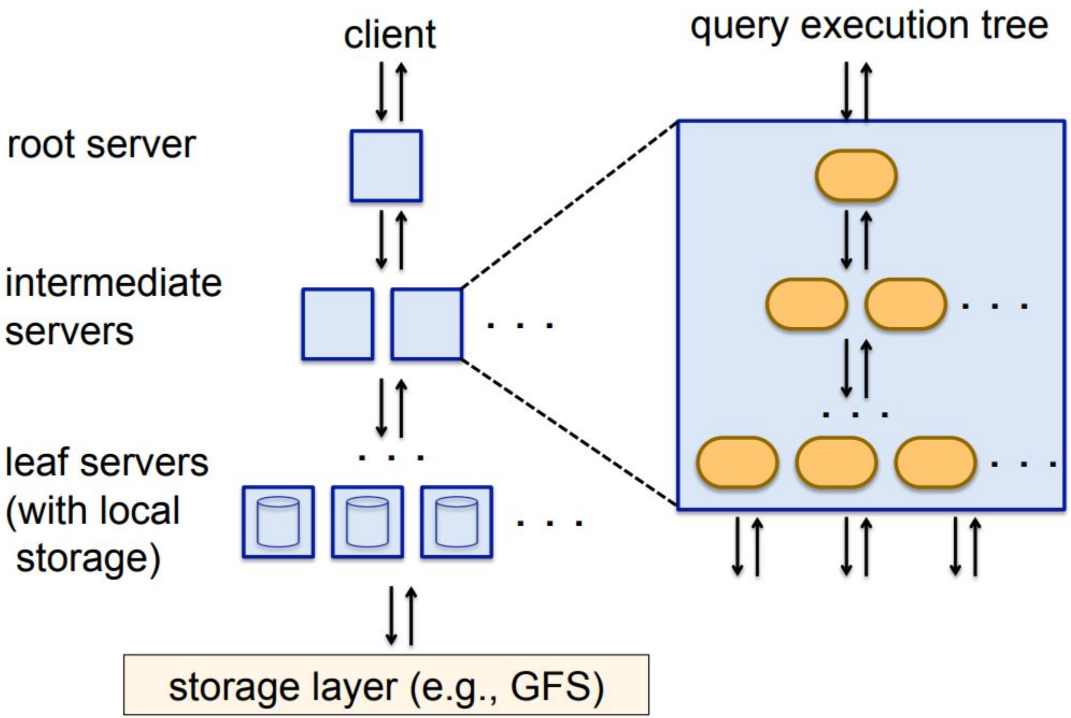
Query Execution

Query Execution

For Query Execution, Dremel uses:

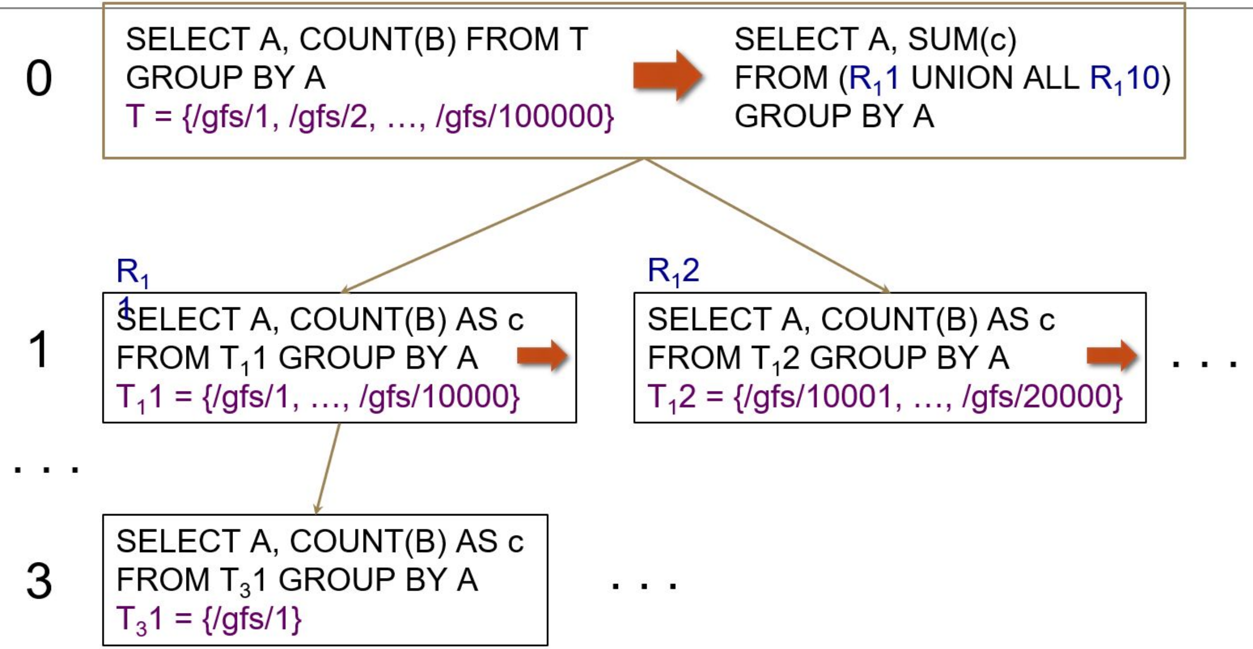
- Tree Architecture:
    - Multi-level Serving Tree
- Query dispatcher
    - Schedule Queries
    - Provide Fault tolerance

System Architecture - Serving Tree



Query Processing Example

Example: count()



Discussion - Pairs

- An observation made was that “If trading speed against accuracy is acceptable, a query can be terminated much earlier and yet see most of the data.”
- In what use cases might it be acceptable to trade speed in exchange for incomplete or inaccurate data results? How often do these use cases come up?

Dremel: A Decade of Interactive SQL Analysis at Web Scale

Dremel:

Dremel’s Key ideas and Architectural Principles:

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

SQL

SQL at Google- Departure from SQL

Google is an earlier pioneer of the Big Data Era.

- **In early 2000s**, developed ethos around distributed infrastructure.
- Conventional Wisdom at Google: “SQL doesn’t scale”

Results in moving away from SQL almost Entirely.

- Solved scalability in exchange of ease of use and ability to iterate quickly.

SQL at Google- Reintroduction

Dremel help reintroduce SQL for big data analysis

- Enabled simple SQL query for analyzing web scale datasets.
- F1 and other OLTP has help return from NoSQL back to SQL.

New Challenge: Many different SQL implementations and Dialects.

Solution: Unifying into one new SQL implementation shared across all SQL-like system, resulting in a new SQL dialect.

Remaining issue: Lack of portability remains industry wide challenge.

SQL - Open Source world

Experienced similar journey of moving away and back into SQL.

- With similar scale and cost challenges with increasing data sizes.
- Experiences same challenges of complexity and slow iteration after migration.
- Similarly pivoted back to SQL (ie. HiveSQL, SparkSQL, and Presto)

Discussion - Groups of 4

- As we have seen with previous works and now with Dremel, there is often a pattern of “forgetting and remembering” when there is a shift for new technologies.
- What strategies can be employed by researchers and practitioners so that new technologies build upon the foundation of previous research rather than disregarding the lessons from it?

Disaggregation

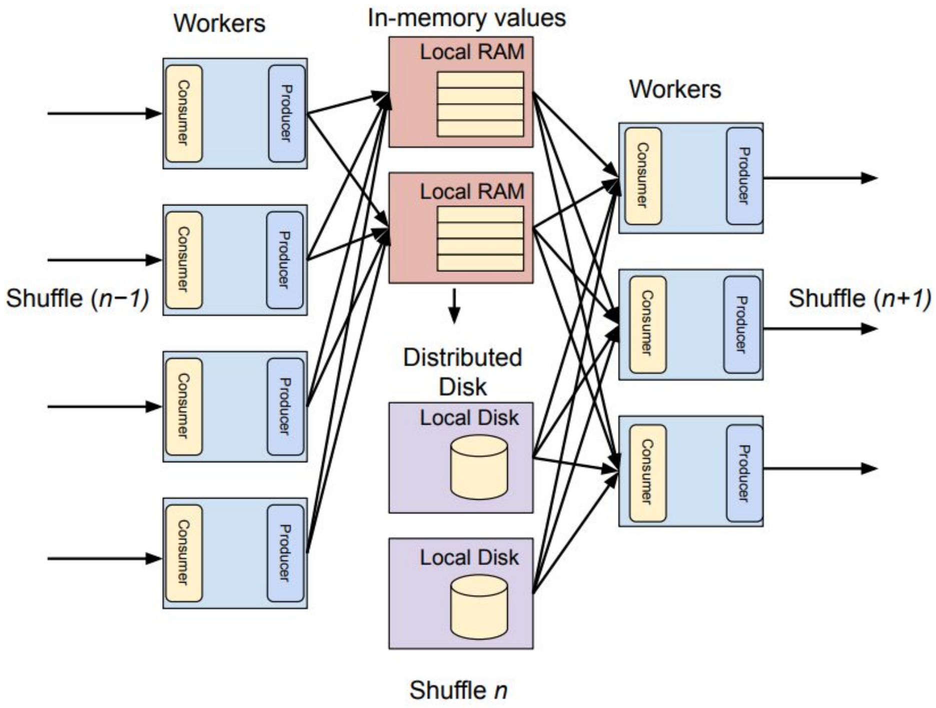
Disaggregated Storage

Dremel initially uses shared nothing servers with directly attached disks.

- As Dremel workloads grew, harder to manage small fleet of dedicated servers.
- Shifted to Borg and cluster management for scalability.
- Exposed to challenges of shared resources.
- Migrated to GFS, disaggregated storage outperformed local disk based system in latency and throughput.

Disaggregated Memory

- Local RAM and disk for storage of sorted intermediate result hit scalability overhead.
- Thus build disaggregated shuffle infrastructure. RAM and storage managed separately. Allow for in memory query execution.
- Significant architectural impact.



Disaggregation Trend

Disaggregation has been proven a major trend in data management.

Ensure better cost-performance and elasticity.

Aspects of Disaggregation:

- Economies of Scale
- Universality
- High level APIs
- Value added packaging

In Situ Data Analysis

In Situ Data Analysis - And Dremel

Refers to accessing data in original place, without upfront data loading and transformation steps.

- Dremel initially designed are reminiscent of traditional DBMS.
  - Explicitly data loading required, proprietary format, inaccessible to other tools.
- As part of Migration to GFS, “open sourced” storage format.
  - Columnar and self describing.
- Allow for interoperation between custom tools and SQL based analysis.

In Situ Data Analysis Evolution

Overtime in situ approach evolved.

- Added file formats.
  - Record based format: Avro, CSV, and JSON
  - Expanded range of data users can query
  - Costed increased I/O Latency
  - Found tradeoff acceptable
- Federation
  - Can do in situ analysis on other systems
  - Allow taken advantage of unique strengths of other systems

In Situ Data Analysis Drawbacks

In Situ Data Analysis like most systems has drawbacks.

- User need to self manage data safely and securely
  - May not be desirable to all users
  - May not be capable of doing it safely and securely
- No opportunity to optimize storage layout or compute statistics.
  - Makes many standard optimization impossible

Serverless Computing

Serverless Computing

Serverless Computing - Elastic, multi-tenant, and on-demand service.

Enabled by the following core ideas:

- Disaggregation
  - Computed, storage and memory, allow for on-demand scaling and compute sharing.
- Fault Tolerance and Restartability
  - Computing resources may be slow or unavailable and thus workers are inherently unreliable.
  - Allow for easy resource adjustments.
- Virtual Scheduling Units
  - Abstract units of compute and memory.

Evolution of Serverless Computing

Dremel Continues to change and evolve its serverless capabilities, some of those ideas are:

- Centralized Scheduling
  - Allow for more fine-grained resource allocation and reservations.
- Shuffle Persistence Layer
  - Allow decoupling scheduling and execution of different stages of the query and for dynamic preemptions.
- Dynamic Query Execution
  - Dynamically adjust query execution plan during runtime.

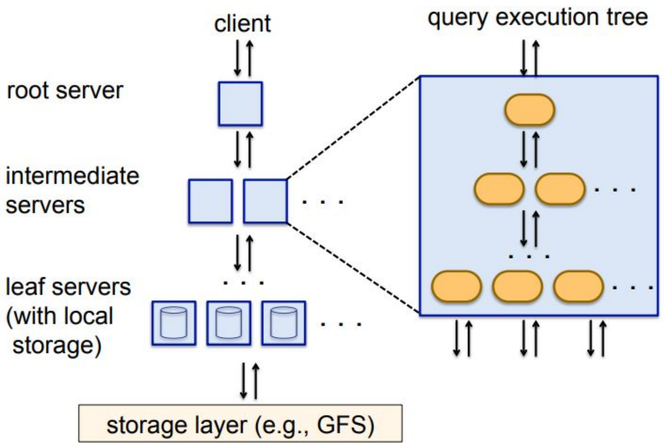
Evolution of Serverless Computing

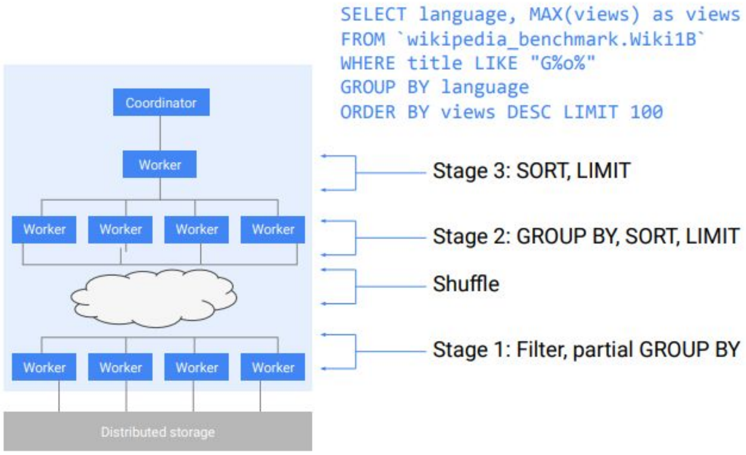
- Centralized Scheduling
    - Allow for more fine-grained resource allocation.
    - Allow for reservations.
    - Uses entire cluster states for better utilization and isolation.

- Shuffle Persistence Layer
    - Allow decoupling scheduling and execution of different stages of the query.
    - Allow for dynamic preemptions

Evolution of Serverless Computing

- Flexible Execution DAGs
  - Fixed tree not ideal for complex query plans.
  - Query coordinator first receive query then orchestrate query execution.
  - Workers allocated as a pool without predefined structure.





Evolution of Serverless Computing

- Dynamic Query Execution
  - Difficult to obtain accurate carbonality estimate during query planning.
  - Thus Dremel enables query execution plans to dynamically change during runtime.
  - Can use statistics collected during query execution.
  - Made possible with shuffle persistence layer and centralized query orchestration.

Discussion: Groups of 4

- How do the trade-offs between security, privacy, operational complexity, performance scalability, and cost impact a company's decision to adopt a serverless DBMS architecture like Dremel, as opposed to managing their own databases?

Columnar Storage

Columnar Storage for nested data

In Early 2000s - Many new applications writes Semistructured data with flexible schemas instead of relational schema.

- Dremel spearheaded use of columnar storage for semistructured data.
- Developments of many open source columnar formats for nested data follows. (Parquet file format, ORC, and Apache Arrows)
- Formats all supports nested and repeated data but are done differently. All with different tradeoffs.

Columnar Storage Example

```
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'
```

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

Dremel

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

ORC

Improved Columnar Format - Capacitor

Allow for:

- Efficient filtering
  - Partition and predicate pruning
  - Vectorization
  - Skip indexes
  - Predicate reordering
- Ability to reorder rows
- Support for more complex schemas

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

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

Interactive Query Latency over Big Data

Query Latency

- Decision decisions of disaggregation, in situ processing and serverless work against interactive query latency.
- Dremel uses additional latency reducing techniques:
  - Stand-by server pool
  - Speculative execution
  - Multi-level execution trees
  - Column-oriented schema representation
  - Balancing CPU and IO with lightweight compression
  - Approximate results
  - Query latency tiers
  - Reuse of file operations
  - Guaranteed capacity
  - Adaptive query scaling

Discussion - Groups of 3

- “Table row-store is a legacy paradigm, the future is columnar-store!”
- Do you agree with this idea? Under what circumstances might one be preferred over the other?

Conclusion

What Dremel got right:

- Disaggregate compute and storage
- On-demand serverless execution
- Columnar storage for nested, repeated and semistructured data
- In situ data analysis

Few things missed or got wrong:

- Missing reliable and fast shuffle layer
- Overlooking importance of managed storage option in addition to in situ analysis
- Need for SQL language standards