Large-scale data analysis at cloud scale

johnwilkes@google.com
2016-09

Based on slides from Eric Schmidt, Greg DeMichillie, Frances Perry, Tyler Akidau, Dan Halperin.
GCP Big Data reference architecture
One possible flow

Application logs

Cloud Pub/Sub
Ingest and propagate

Cloud Dataflow
Processing and imperative analysis

BigQuery
Durable storage and interactive analysis

Questions you might ask
- Which application sections are receiving the most impressions?
- Who is the top artist by stream starts in the last minute?
- What is the average session length for users in Seattle?
10+ years of tackling Big Data problems

Google Papers

- GFS
- Map Reduce
- BigTable
- Dremel
- PubSub
- Flume Java
- Millwheel
- Dataflow
- Tensorflow
10+ years of tackling Big Data problems

Google Papers
- GFS
- Map Reduce
- BigTable
- Dremel
- PubSub
- Flume Java
- Millwheel
- Dataflow
- Tensorflow
The Dremel paper

Dremel: Interactive Analysis of Web-Scale Datasets

Sergey Melnik, Andrey Gubarev, Jing Jing Long, Geoffrey Romer, Shiva Shivakumar, Matt Tolton, Theo Vassilakis
Google, Inc.
{melnik,andrey,jlong,gromer,shiva,mtolton,theov}@google.com

ABSTRACT
Dremel is a scalable, interactive ad-hoc query system for analysis of read-only nested data. By combining multi-level execution trees and columnar data layout, it is capable of running aggregation queries over trillion-row tables in seconds. The system scales to thousands of CPUs and petabytes of data, and has thousands of users at Google. In this paper, we describe the architecture and implementation of Dremel, and explain how it complements MapReduce-based computing. We present a novel columnar storage representation for nested records and discuss experiments on few-thousand node instances of the system.

exchanged by distributed systems, structured documents, etc. lend themselves naturally to a nested representation. Normalizing and recombining such data at web scale is usually prohibitive. A nested data model underlies most of structured data processing at Google [21] and reportedly at other major web companies.

This paper describes a system called Dremel[1] that supports interactive analysis of very large datasets over shared clusters of commodity machines. Unlike traditional databases, it is capable of operating on in situ nested data. In situ refers to the ability to access data ‘in place’, e.g., in a distributed file system (like GFS [14]) or another storage layer (e.g., Bigtable [8]). Dremel can execute many queries over such data that would ordinarily require a sequence of

Dremel in 2016

Dremel is **mission critical** for Google

In production for 10+ years, in every Google datacenter

Internal usage every month:

- $O(\text{Exabytes})$ analyzed
- $O(\text{Quadrillions})$ of rows scanned
- 80% of Googlers use it
BigQuery ≈ Dremel + cloud

Our idea of highly available, managed analytics:

- no indexing, no resharding, no storage resizing
- just ...

RUN QUERY
BigQuery ≈ Dremel + cloud

- Full-managed data warehouse
- Fast, petabyte-scale with SQL interface
- Encrypted, durable and highly available
- Near-unlimited resources, only pay for what you use
- Streaming ingest for unbounded data sets

- General Availability in 2012
- Same engineering team
- Same code base
But what have we done lately?
BigQuery: 5+ years of innovation

- 2010: Beta Release at Google I/O
- 2012: Public launch
- 2013: Big JOIN support
- 2014: Large query results
- 2015: Faster shuffle, Capacitor
- 2016: 100k qps streaming, Dremel X, User-defined functions, Dynamic Execution

---

Google Cloud Platform
Dremel architecture: 2006–2015

- Long-lived Serving Tree
- Partial Reduction
- Diskless data flow
- Columnar Storage

Distributed Storage

Mixer 0

Mixer 1

Leaf

Leaf

Leaf

Leaf

Dremel X architecture (2015–now)

- **Master**
- **Distributed Storage**
  - **Shard**
  - **Dynamic Serving Tree**
  - **Columnar Storage**

SELECT play_count FROM songs WHERE name CONTAINS "Sun";

<table>
<thead>
<tr>
<th>Data</th>
<th>Decompress</th>
<th>Filter</th>
<th>Emit</th>
</tr>
</thead>
<tbody>
<tr>
<td>F$h5</td>
<td>xc*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rm7y5</td>
<td>c8!</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rm7y5</td>
<td>8ec</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F$h5</td>
<td>7h!</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4t#h</td>
<td>a7c</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rm7y5</td>
<td>c-%</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

{Hey Jude, 5375}  CONTAINS("Sun")  {7833}
{My Michelle, 2188}  CONTAINS("Sun")
{My Michelle, 9363}  CONTAINS("Sun")
{Hey Jude, 9502}  CONTAINS("Sun")
{Here Comes The Sun, 7383}  CONTAINS("Sun")
{My Michelle, 3912}  CONTAINS("Sun")
**Storage engine: Capacitor (2016–now)**

```
SELECT play_count FROM songs WHERE name CONTAINS "Sun";
```

<table>
<thead>
<tr>
<th>Data</th>
<th>Emit</th>
<th>Dictionary</th>
<th>Filter</th>
<th>Lookup</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>xc*</td>
<td>0</td>
<td>CONTAINS(&quot;Sun&quot;)</td>
<td>F</td>
</tr>
<tr>
<td>1</td>
<td>c8!</td>
<td>1</td>
<td>CONTAINS(&quot;Sun&quot;)</td>
<td>F</td>
</tr>
<tr>
<td>1</td>
<td>8ec</td>
<td>2</td>
<td>CONTAINS(&quot;Sun&quot;)</td>
<td>T</td>
</tr>
<tr>
<td>2</td>
<td>7h!</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>a7c</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>c-%</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Performance improvement:
- 2x faster average over all queries
- 10–1000x faster for selective filters
Cloud Dataflow
One possible flow

Application logs

Cloud Pub/Sub
*Ingest and propagate*

Cloud Dataflow
*Processing and imperative analysis*

BigQuery
*Durable storage and interactive analysis*

Questions you might ask
- Which application sections are receiving the most impressions?
- Who is the top artist by stream starts in the last minute?
- What is the average session length for users in Seattle?
Time-to-answers matters

- Sequence a human genome
- Process point of sale transaction logs
- Who is the best at collecting Poké Balls?
- Who/what is trending?
Data boundedness

**Bounded**
- Finite data set
- Fixed schema
- Complete
- Typically at rest

**Unbounded**
- Infinite data set
- Potentially changing schema
- Never complete
- Typically not at rest

Time
Latency happens

**Transmit**
- Network delay or unavailable
- Ingest delay (write latency)
- Ingest failure

**Ingest**
- Throughput (read latency)
- Backlog
- Hardware failure

**Process**
- Poor engine design
- Starved resources
- Internal backlog
- Confused heuristics
The evolution of Apache Beam

MapReduce

Apache Beam

Google

Google Cloud Dataflow

Apache

Colossus
BigTable
PubSub
Dremel
Spanner
Megastore
Millwheel
Flume

Map
Base
Hadoop
Drill
Spark
Hive
Dremel
BigTable
PubSub
Dremel
Apache Beam (incubating) is a unified programming model designed to provide efficient and portable data processing pipelines.
1. Classic batch

2. Batch with fixed windows

3. Streaming

4. Streaming with speculative + late data

5. Streaming with retractions

6. Sessions

The Apache Incubator Project
http://incubator.apache.org/
Formalizing Event-Time Skew

Watermarks describe event time progress in the processing time space:

"No timestamp earlier than the watermark will be seen"

Often based on heuristics

- Too slow? Results are delayed :-(
- Too fast? Some data is late :-(

The Apache Incubator Project
http://incubator.apache.org/
● **What** are you computing?

● **Where** in event time?

● **When** in processing time?

● **How** do refinements relate?
What are you computing?

Per-element

Aggregations

Compositions (subgraphs)
What: computing integer sums
What: computing integer sums
Windowing divides data into event-time-based finite chunks.

**Where in event time?**

- **Fixed**
- **Sliding**
- **Sessions**

Often required when doing aggregations over unbounded data.
Where: Fixed 2-minute Windows
When in processing time?

- Triggers control when results are emitted.
- Triggers are often relative to the watermark.
When: triggering at the watermark

Perfect Watermark

Heuristic Watermark

Perfect watermark:
Ideal watermark:

Heuristic watermark:
Ideal watermark:
When: early and late firings
How do refinements relate?

Q: if there are multiple panes per window ... what should be emitted?

- **discard** – emit the delta over the pane
- **accumulate** – emit running total
- **accumulate + retract** – retract last sum & emit new running total

A: drive by needs of the downstream consumer

*Accumulating & Retracting not yet implemented in Apache Beam.*
How: add newest, remove previous
Customizing What When Where How

1. Classic batch
2. Batch with fixed windows
3. Streaming
4. Streaming with speculative + late data
5. Streaming with retractions
The straggler problem

Work is unevenly distributed across tasks

• Underlying data size
• Processing differences
• Runtime effects

Effects are cumulative per stage
**Beam readers enable dynamic adaptation**

**Beam readers** provide simple progress signals, enable runners to take action based on execution-time characteristics.

APIs for how much work is pending:
- Bounded: `double getFractionConsumed()`
- Unbounded: `long getBacklogBytes()`

Work-stealing:
- Bounded: `Source splitAtFraction(double)`
  
  `int getParallelismRemaining()`
Dynamic work rebalancing

- **Done work**
- **Active work**
- **Predicted completion**

Tasks

Time

- **Now**
- **Predicted mean time**

The Apache Incubator Project
http://incubator.apache.org/
Dynamic work rebalancing

- **Done work**
- **Active work**
- **Predicted completion**
Dynamic work rebalancing: a real example

Beam pipeline on the Google Cloud Dataflow runner

2-stage pipeline, split “evenly” but uneven in practice

Same pipeline with dynamic work rebalancing
Initially allocate ~80 workers based on input size

Multiple rounds of upsizing enabled by dynamic splitting

Scale up to 1000 workers * tasks stay well-balanced * without initial oversplitting

Long-running tasks aborted without causing stragglers

Beam pipeline on the Google Cloud Dataflow runner

Dynamic work rebalancing + Autoscaling
Cloud Dataflow execution runner

- Graph optimization
- Work scheduler
- Resource auto-scaler
- Dynamic work rebalancer
- Compute and Storage
- Resource management
- Intelligent watermarking
- Auto-healing
- Monitoring
- Log collection

SOURCE
- Unbounded
- Bounded

SINK
The Dataflow Model & Cloud Dataflow

**Dataflow Model & SDKs**
- a unified model for batch and stream processing

**Google Cloud Dataflow**
- no-ops, fully managed service
The **Beam** Model & Cloud Dataflow

**Apache Beam**

a unified model for batch and stream processing

*supporting multiple runtimes*

**Google Cloud Dataflow**

*a great place to run Beam*
What is in Apache Beam?

1. The Beam Model: **What** / **Where** / **When** / **How**

2. SDKs for writing Beam pipelines – starting with Java

3. Runners for existing distributed processing backends
   - **Apache Flink** (thanks to data Artisans)
   - **Apache Spark** (thanks to Cloudera)
   - **Google Cloud Dataflow** (fully managed service)
   - **Local** (in-process) runner for testing
## Categorizing runner capabilities

### What is being computed?

<table>
<thead>
<tr>
<th></th>
<th>Beam Model</th>
<th>Cloud Dataflow</th>
<th>Apache Flink</th>
<th>Apache Spark</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParDo</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GroupByKey</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>~</td>
</tr>
<tr>
<td>Flatten</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Combine</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Composite Transforms</td>
<td>✓</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>Side Inputs</td>
<td>✓</td>
<td>✓</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>Source API</td>
<td>✓</td>
<td>✓</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>Aggregators</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>Keyed State</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
</tbody>
</table>

### Where in event time?

<table>
<thead>
<tr>
<th></th>
<th>Beam Model</th>
<th>Cloud Dataflow</th>
<th>Apache Flink</th>
<th>Apache Spark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global windows</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Fixed windows</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>Sliding windows</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>Session windows</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>Custom windows</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>Custom merging windows</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>Timestamp control</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
</tr>
</tbody>
</table>

### When in processing time?

<table>
<thead>
<tr>
<th></th>
<th>Beam Model</th>
<th>Cloud Dataflow</th>
<th>Apache Flink</th>
<th>Apache Spark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configurable triggering</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>Event-time triggers</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>Processing-time triggers</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>Count triggers</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>[Meta]data driven triggers</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>Composite triggers</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>Allowed fastness</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>Timers</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
</tbody>
</table>

### How do refinements relate?

<table>
<thead>
<tr>
<th></th>
<th>Beam Model</th>
<th>Cloud Dataflow</th>
<th>Apache Flink</th>
<th>Apache Spark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Discarding</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Accumulating</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>Accumulating &amp; Retracting</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
</tbody>
</table>

Apache Beam roadmap

02/01/2016
Enter Apache Incubator

02/25/2016
1st commit to ASF repository

1H 2016
Refactoring (slight chaos)

2H 2016
Portability basics + Multiple runners + Community growth
⇒ towards a Top-Level Project

1H 2017
Stable APIs
Growing the Beam community

**Collaborate** – a community-driven effort across many organizations and contributors

**Grow** – the Beam ecosystem and community with active, open involvement
Now

1st Wave
Colocation

2nd Wave
Virtualized
Datacenters

User Managed
User Configured
User Maintained

Next

Intelligent Services
Auto Everything
Big Data with Google

Focus on insights, not infrastructure

From batch to real-time

Apache Beam offers a common model across execution engines: please contribute!

johnwilkes@google.com