Agenda

1. Google’s Data Processing Story
2. Philosophy of the Beam programming model
3. Apache Beam project
Google’s Data Processing Story
Data Processing @ Google

- MapReduce
- GFS
- Big Table
- Dremel
- FlumeJava
- Pregel
- Colossus
- Spanner
- MillWheel

Timeline:
- 2002: GFS
- 2004: Big Table
- 2006: Dremel
- 2008: Pregel
- 2010: FlumeJava
- 2012: Colossus
- 2014: Spanner
- 2016: MillWheel

Dataflow
MapReduce: SELECT + GROUP BY

(distributed input dataset)

Map (SELECT)

Shuffle (GROUP BY)

Reduce (SELECT)

(distributed output dataset)
Data Processing @ Google

- **2002**: GFS
- **2004**: Big Table
- **2006**: Dremel
- **2008**: Pregel
- **2010**: FlumeJava
- **2012**: Spanner
- **2014**: Colossus
- **2016**: MillWheel

**Dataflow**
FlumeJava Pipelines

- A Pipeline represents a graph of data processing transformations
- PCollections flow through the pipeline
- Optimized and executed as a unit for efficiency
Example: Computing mean temperature

// Collection of raw events
PCollection<SensorEvent> raw = ...;

// Element-wise extract location/temperature pairs
PCollection<KV<String, Double>> input =
    raw.apply(ParDo.of(new ParseFn()))

// Composite transformation containing an aggregation
PCollection<KV<String, Double>> output = input
    .apply(Mean.<Double>perKey());

// Write output
output.apply(BigtableIO.Write.to(...));
So, people used FJ to process data...
...big data...
...really, really big...
Batch failure mode #1

Latency
Batch failure mode #2: Sessions

Tuesday: Jose, Lisa, Ingo, Asha, Cheryl, Ari

Wednesday: Jose, Lisa, Ingo, Asha, Cheryl, Ari

MapReduce
State of the art until recently: Lambda Architecture

Historical events → Periodic batch processing → Exact historical model

Stream processing system → Continuous updates → Approximate real-time model
Data Processing @ Google

- MapReduce
- GFS
- Big Table
- Dremel
- Pregel
- FlumeJava
- Colossus
- Spanner
- MillWheel

Timeline:
- 2002: GFS
- 2004: Big Table
- 2006: Dremel
- 2008: Pregel
- 2010: FlumeJava
- 2012: Colossus
- 2014: Spanner
- 2016: MillWheel

Dataflow
MillWheel: Deterministic, low-latency streaming

- Framework for building low-latency data-processing applications
- User provides a DAG of computations to be performed
- System manages state and persistent flow of elements
Streaming or Batch?

1 + 1 = 2

Correctness

Latency

Why not both?
What are you computing?

Where in event time?

When in processing time?

How do refinements relate?
Where in event time?

- **Windowing** divides data into event-time-based finite chunks.

- Required when doing aggregations over unbounded data.
When in Processing Time?

- Triggers control when results are emitted.
- Triggers are often relative to the watermark.
How do refinements relate?

PCollection<KV<String, Integer>> output = input
   .apply(Window.into(Sessions.withGapDuration(Minutes(1)))
      .trigger(AtWatermark() 
         .withEarlyFirings(AtPeriod(Minutes(1))) 
         .withLateFirings(AtCount(1)))
      .accumulatingAndRetracting())
   .apply(new Sum());
Data Processing @ Google

- MapReduce
- GFS
- Big Table
- Dremel
- Pregel
- FlumeJava
- Colossus
- Spanner
- MillWheel
- Dataflow

- 2002
- 2004
- 2006
- 2008
- 2010
- 2012
- 2014
- 2016

Dataflow
Google Cloud Dataflow

A fully-managed cloud service and programming model for batch and streaming big data processing.
Dataflow SDK

- **Portable API** to construct and run a pipeline.
- Available in Java and Python (alpha)
- Pipelines can run…
  - On your development machine
  - On the **Dataflow Service** on **Google Cloud Platform**
  - On third party environments like Spark or Flink.
Dataflow ⇒ Apache Beam
Pipeline p = Pipeline.create(options);

p.apply(TextIO.Read.from("gs://dataflow-samples/shakespeare/*"))
  .apply(FlatMapElements.via(
      word → Arrays.asList(word.split("[^a-zA-Z']\+")))
  .apply(Filter.byPredicate(word → !word.isEmpty()))
  .apply(Count.perElement())
  .apply(MapElements.via(
      count → count.getKey() + ": " + count.getValue())
  .apply(TextIO.Write.to("gs://.../...")))
Apache Beam ecosystem

- End-user's pipeline
- Libraries: transforms, sources/sinks etc.
- Language-specific SDK
- Beam model (ParDo, GBK, Windowing...)
- Runner
- Execution environment

Execution environments:
- Java
- Python
- ...

Supporting frameworks:
- Spark
Apache Beam ecosystem

- End-user's pipeline
- Libraries: transforms, sources/sinks etc.
- Language-specific SDK
- Beam model (ParDo, GBK, Windowing...)
- Runner
- Execution environment

Execution environment (Java, Python, ...)

Spark
Apache Beam Roadmap

- **02/01/2016**: Enter Apache Incubator
- **02/25/2016**: 1st commit to ASF repository
- **Early 2016**: Design for use cases, begin refactoring
- **Mid 2016**: Slight chaos
- **Late 2016**: Multiple runners execute Beam pipelines
## Runner capability matrix

### 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>X</td>
<td>X</td>
<td>X</td>
<td>X</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>X</td>
</tr>
<tr>
<td>Custom windows</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td>Custom merging windows</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td>Timestamp control</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</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 liveness</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>
• **Multiple SDKs** with shared pipeline representation

• **Language-agnostic runners** implementing the model

• **Fn Runners** run language-specific code
Recap: Timeline of ideas

2004  **MapReduce** (SELECT / GROUP BY)
Library > DSL
Abstract away fault tolerance & distribution

2010  **FlumeJava**: High-level API (typed DAG)

2013  **MillWheel**: Deterministic stream processing

2015  **Dataflow**: Unified batch/streaming model
Windowing, Triggers, Retractions

2016  **Beam**: Portable programming model
Language-agnostic runners
Learn More!

**Programming model**
The World Beyond Batch: [Streaming 101](#), [Streaming 102](#)
The Dataflow Model paper

**Cloud Dataflow**
http://cloud.google.com/dataflow/

**Apache Beam**
https://wiki.apache.org/incubator/BeamProposal
http://beam.incubator.apache.org/
Dataflow/Beam vs. Spark
Thank you