An introduction to Apache Beam

For streaming analytics

FOSDEM 2023
Brussels, Feb 4th
Israel Herraiz

@herraiz

Strategic Cloud Engineer

Google Cloud
Apache Beam
What is Apache Beam?

- Batch Processing
- Streaming Processing
- Beam Model
- Language SDKs
What is Apache Beam?

**SDKs**
- Java
- Python
- Scala
- Golang

**Runners**
- Direct Runner
- Dataflow
- Flink
- Hazelcast JET
- Spark
- Nemo
- Samza
- Twister2
The problem with streaming
Data streams: unbounded data sources
Arrival out of order
Micro-batching does not solve the problem with out of order
In any data processing system:

- There is a certain amount of lag between:
  - The **event time**, when a data event occurs (determined by the timestamp on the data element itself).
  - The **processing time**, when a data element gets processed at any stage in a pipeline (determined by the clock on the processing system).

- There are no guarantees that data events will appear in a pipeline in the same order that they were generated.
In other words: event time vs. processing time in Star Wars

### Event Time

<table>
<thead>
<tr>
<th>Episode IV</th>
<th>Episode V</th>
<th>Episode VI</th>
<th>Episode I</th>
<th>Episode II</th>
<th>Episode III</th>
<th>Episode VII</th>
<th>Episode VIII</th>
<th>Episode IX</th>
</tr>
</thead>
<tbody>
<tr>
<td>A new Hope</td>
<td>The Empire</td>
<td>Return of the</td>
<td>The Phantom</td>
<td>Attack of The</td>
<td>Revenge of the</td>
<td>The Force</td>
<td>The Last Jedi</td>
<td>The Rise of</td>
</tr>
<tr>
<td></td>
<td>Strikes Back</td>
<td>Jedi</td>
<td>Menace</td>
<td>Clones</td>
<td>Sith</td>
<td>Awakens</td>
<td>Jedi</td>
<td>Skywalker</td>
</tr>
</tbody>
</table>

### Processing Time

Source: *Introduction to Apache Flink* by Ellen Friedman, Kostas Tzoumas
Dealing with out of order: Windows
Answering four questions

What results are calculated?

Where in event time are results calculated?

When in processing time are results materialized?

How do refinements of results relate?
The Beam Model: **What** is Being Computed?

```java
PCollection<KV<String, Integer>> scores = input
    .apply(Sum.integersPerKey());
```
The Beam Model: **What** is Being Computed?
PCollection<KV<String, Integer>> scores = input
   .apply(Window.into(FixedWindows.of(Duration.standardMinutes(2))))
   .apply(Sum.integersPerKey());
The Beam Model: *Where* in event time?
The Beam Model: **When** in processing time?

```java
PCollection<KV<String, Integer>> scores = input
    .apply(Window.into(FixedWindows.of(Duration.standardMinutes(2))
        .triggering(AtWatermark()))
    .apply(Sum.integersPerKey());
```
The Beam Model: **When** in processing time?
The Beam Model: **How** do we refine/recalculate?

```java
PCollection<KV<String, Integer>> scores = input
    .apply(Window.into(FixedWindows.of(Duration.standardMinutes(2))
        .triggering(AtWatermark()
            .withEarlyFirings(AtPeriod(Duration.standardMinutes(1)))
            .withLateFirings(AtCount(1)))
        .accumulatingFiredPanes())
    .apply(Sum.integersPerKey());
```
The Beam Model: How do we refine/recalculate?
beam.apache.org/get-started/mobile-gaming-example/
Dealing with out of order: Stateful functions
Stateful processing

Persistent mutable state

It is partitioned by key and window

It can be read and written during the processing of each element

The input needs to be a PCollection of KV

ParDo

Persistent mutable state
State variables and timers: example

ParDo

@ProcessElement

@OnTimer

call back when data is stale
call back at expiry

External Service

State

buffer

count

Event-time

Processing-time
State & timers patterns:
youtube.com/watch?v=RQjJ0BDKI_k

Using the Timer &State API to solve times series use cases
youtube.com/watch?v=Q_v5Zsjuuzg
Other goodies in streaming:
ML inference at scale
ML inference

Local and remote inference

Resource hints: use GPU, specify memory requirements

```python
with pipeline as p:
    data = p | 'Read' >> beam.ReadFromSource('a_source')

model_a_predictions = data | RunInference(<model_handler_A>).with_resourceHints(min_ram="20GB")
model_b_predictions = model_a_predictions | beam.Map(x) | RunInference(<model_handler_B>)
    .with_resource_hints( min_ram="4GB", accelerator="type:nvidia-tesla-k80;count:1;install-nvidia-driver")
```

[beam.apache.org/documentation/sdks/python-machine-learning/](beam.apache.org/documentation/sdks/python-machine-learning/)
[beam.apache.org/documentation/ml/overview/](beam.apache.org/documentation/ml/overview/)
Other goodies in streaming:
In Java (or lang of choice) too!
Cross language transforms: use any transform from any SDK in any other SDK.

For instance, RunInference in Java


[beam.apache.org/documentation/programming-guide/#multi-language-pipelines](beam.apache.org/documentation/programming-guide/#multi-language-pipelines)
Recommended readings and links
The book on Streaming Systems

Tyler Akidau, Slava Chernyak, Reuven Lax

http://streamingsystems.net/

"If you care about the correctness of your streaming and batch processing jobs, this book is a must-read. It provides the most clear-thinking and logical discussion of the topic that I have seen, and its ideas are brilliantly explained."

—Martin Kleppmann
University of Cambridge