Detecting Change-Points in Apache Beam

By Devon Peticolas - Oden Technologies

https://github.com/x/slides/tree/master/beam-summit-2022
Devon Peticolas

Principal Engineer @ Oden Technologies
“Beam Guy”
In This Talk

- What is Change-Point Detection?
- Why and how does Oden use Change-Point Detection to deliver features?
- Methods of doing Change-Point Detection in Beam.
- Methods of doing Change-Point Detection with Smoothing.
- Impacts of event sparsity, lateness, and order.
A little about Oden
Oden’s Customers

Medium to large manufacturers in plastics extrusion, injection molding, and pipes, chemical, paper and pulp.

Process and Quality Engineers looking to centralize, analyze, and act on their data.

Plant managers who are looking to optimize logistics, output, and cost.
Interactive Time-series Analysis

- Compare performance across different equipment.
- Visualize hourly uptime and key custom metrics.
- Calculations for analyzing and optimizing factory performance.
Real Time Manufacturing Data

- Streaming second-by-second metrics
- Interactive app that prompts on production state changes and collects user input.
Background: How Oden Uses Beam
How Oden Uses Beam

- Ingesting “raw” manufacturing data and mapping it into Oden “events”
- Combining events using streaming joins
- Making customer-configured transformations to events
- Transforming metric events into contextual interval events

* Lots of Side-Input Joining
* Lots of Complex Windowing
* Lots of Performance Concerns
Streaming Factory Data - In Summary

- PLC
- Raw Factory Data
- Transform Factory Data
  - Metrics
    - TSDB
  - Intervals
    - Postgres
Streaming Factory Data - In Summary

PLC

Acquisition

Raw Factory Data

Flaming Rainbow Bridge to Asgard

Everytime I remember this exists I get sad

We built this to close a deal and now I’m not sure if we can kill it

Calculated Metrics

Rollups (600s)

Postgres

Windowed Calculated Metrics (60s)

State Change Detection

API that uses regex as a language parser with terrifying effectiveness

Also Metrics

Probably a waste of money

Also BigQuery because we don’t trust our DS team with either TSDB

Definitely a waste of money

New TSDB

TSDB

We built this to close a deal and now I’m not sure if we can kill it

Google-provided DataFlow Template that DROPs DATA when redeployed

Also Metrics

Definitely a waste of money

11
Data is Grouped by Production Line

- Downtime
- Uptime
- Disabled After Tool Change: 366.0 ft ...
- D.
- Disabled After...
- N.
- F...
- No W...
- N...
- F N...

17311

Graph showing:
- Line Speed
- Block Temperature
- Mantle Output Power

Austin, 2022
Metrics

Each metric describes a sensor
Each value is a float
One value per second per metric
Intervals

Each interval describes the state of a manufacturing line.

Each value is categorical string.

One value per change.
Creating Intervals from Metrics
Use Case:
Creating Intervals from Metrics
Visualizing Events

![Graph showing a linear relationship between Process Time and Event Time](image)
Visualizing Events

- On-Time
- Late
- Out of Order

Process Time vs. Event Time graph.
Visualizing Events

The Photographer

People in group photo

Process Time

Event Time

Austin, 2022
Visualizing Events

The Photographer

People in group photo
Visualizing Events w/ Change
**Availability** = \( \frac{\text{Uptime}}{(\text{Uptime} + \text{Downtime})} \)

**Overall Equipment Efficiency** = **Availability** \( \times \) **Performance** \( \times \) **Quality**
Metrics into Categorical Values

// Load configuration every 5-minutes.
PCollectionView<Config> configView = p
  .apply(GenerateSequence
           .from(0)
           .withRate(1, Duration.standardMinutes(5)))
  .apply(MapElements(...)) // API call
  .apply(View.asSingleton());

// Map metric values to categorical values using config side-input.
p.apply(ParDo.of(new DoFn<Metric, String>() {
    public void processElement(Metric m, ProcessContext c) {
      Config config = c.sideInput(configView)
      if (m.value > config.forMetric(m)) {
        c.output("up");
      } else {
        c.output("down");
      }
    }
  }).withSideInputs(configView))
<table>
<thead>
<tr>
<th>Line</th>
<th>Line Aggregate State History</th>
<th>Util</th>
<th>Perf</th>
<th>Production</th>
<th>CURRENT RUN ON LINE</th>
<th>Current Speed</th>
<th>Current Production</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compounding 1</td>
<td>Uptime 4h 50m</td>
<td>100%</td>
<td>0%</td>
<td>15,000 lb</td>
<td>UHK-71120000 25h 13m</td>
<td>7,500</td>
<td>174,222 lb</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Product 6D50D9B2 Batch 3C64E95D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Extruder 1</td>
<td>Downtime 2m 55s</td>
<td>81%</td>
<td>181%</td>
<td>36,407 ft</td>
<td>Test Run 347d 1h</td>
<td>0</td>
<td>12,385,545 ft</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Product 109810-P-A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Extruder 2</td>
<td>Uptime 49m 2s</td>
<td>95%</td>
<td>79%</td>
<td>5,066 ft</td>
<td>XYZ-123 656d 9h</td>
<td>50</td>
<td>1,753,394 ft</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Product 85EB50E1 Batch 22CD2E30</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Extruder 3</td>
<td>Uptime 3m 44s</td>
<td>63%</td>
<td>78%</td>
<td>12,468 ft</td>
<td>UHL-64957000 2h 56m</td>
<td>183</td>
<td>17,830 ft</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Product D48199B2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Line 01</td>
<td>Uptime 53m 33s</td>
<td>97%</td>
<td>0%</td>
<td>0 ft</td>
<td>line_demo_in_PROD 53d 12h</td>
<td>330</td>
<td>0 ft</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Product 2AFBCA5E Batch D824D563</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**OEE** 0%  **Utilization** 87.3%  **Performance** 80.6%  **Quality** 0%
Solution:
Using Beam State
Beam State to Detect Change-Points

DoFn<KV<String, T>, T> { 
  StateSpec<ValueSpec<T>> prevSpec = 
    StateSpecs.value(...);

  public void processElement(
    ProcessContext c,
    ValueState<T> prev) {
    T curr = 
      c.element().getValue();
    T last = prev.read();
    if (curr != last) {
      c.output(curr);
    }
  }
}
Issues:
Using Beam State
Beam State to Detect Change-Points

DoFn<KV<String, T>, T> {
  StateSpec<VALUESpec<T>> prevSpec =
  StateSpecs.value(...);

  public void processElement(
    ProcessContext c,
    ValueState<T> prev) {
    T curr =
    c.element().getValue();
    T last = prev.read();
    if (curr != last) {
      c.output(curr);
    }
  }
}
Solution:
Watermark-Triggered Windows
Watermark-Triggered Windows

Window into(
  SlidingWindows
  .of(TWO SECONDS)
  .every(ONE SECOND))
  .accumulatingFiredPanes()
  .triggering(
    Repeatedly.forever(
      AfterWatermark
      .pastEndOfWindow()))

Process Time

Event Time

lag
Watermark-Triggered Windows and Out-of-order Data

Window
  .<T>into(
    SlidingWindows
    .of(TWO SECONDS)
    .every(ONE SECOND))
    .accumulatingFiredPanes()
    .triggering(
      Repeatedly.forever(
        AfterWatermark
        .pastEndOfWindow()))

Process Time

Event Time
Issues:
Watermark-Triggered Windows and Lag
Window
  .<T>into(
      SlidingWindows
        .of(TWO_SECONDS)
        .every(ONE_SECOND))
  .accumulatingFiredPanes()
  .triggering(
        Repeatedly.forever(
            AfterWatermark
                .pastEndOfWindow()))
Non-homogeneous Lag

PLC

Raw Factory Data

Transform Factory Data

TSDB

Metrics

Change Point Detection

Intervals

Postgres
Non-homogeneous Lag

PLC

Raw Factory Data

Transform Factory Data

Metrics

TSDB

PLC

PLC
Window.into(
    SlidingWindows.of(TWO_SECONDS)
    .every(ONE_SECOND)
    .accumulatingFiredPanes()
    .triggering(
        Repeatedly.forever(
            AfterWatermark.pastEndOfWindow()))
)

High watermark caused by bad factory

Slow Detection
Solution:
Data-Triggered Windows
Data-Triggered Windows

Window
  .<T>into(
    SlidingWindows
    .of(TWO SECONDS)
    .every(ONE SECOND))
  .accumulatingFiredPanes()
  .triggering(
    Repeatedly.forever(
      AfterPane
      .elementCountAtLeast(2)))
Data-Triggered Windows

Window
  .<T>into(
    SlidingWindows
    .of(TWO_SECONDS)
    .every(ONE_SECOND))
  .accumulatingFiredPanes()
  .triggering(
    Repeatedly.forever(
      AfterPane
      .elementCountAtLeast(2))))
Issues:
Using Windows but Sparse Data
Watermark-driven trigger

Process Time

Event Time

$> \Delta_{\text{max}}$
Data-driven trigger must trigger every element

\[ > \Delta_{\text{max}} \]
Was there a change at t3?
Use Case: Creating “Smoothed” Intervals from Metrics
This chart illustrates the line speed over time. The key points are:

- **Downtime** is indicated by the brown area.
- **Uptime** is indicated by the green area.
- **Not This** indicates a spike in line speed at 10:55.
- **This** indicates a more stable line speed pattern throughout the observed period from 10:45 to 11:15.

The chart shows a significant spike in line speed at 10:55, labeled as "Not This," which is not desirable. The line speed stabilizes after this peak, indicating a more stable and preferable operational period labeled as "This."
This

Not This

This
Smoothing eliminates the first $n$-seconds. This is the same as our sparse data.
$> \text{smothe} + \Delta_{\text{max}}$
Created lag is always \( \geq \text{smoothe} \)

lag

---

Abbreviation: smoothe
Solution:
State + Sliding Windows
I think this is wrong but...

smooth-time + event-time $\Delta_{\text{max}}$ + out-of-order $\Delta_{\text{max}}$ < process-time $\Delta_{\text{min}}$
Key Takeaways

● Oden Uses Change-Point Detection to transform Metrics into Intervals
● Beam State is fast and good at sparsity, but bad at out-of-order
● Windowing is slower and good at out-of-order, but bad at sparsity
● Combining Beam State and Windowing is good at out-of-order and sparsity
● “Smoothed” Change-Point Detection is just a sparsity problem
Thank You

And a special thanks to Jie Zhang, Jake Skelcy, and Deepak Turaga
Questions?

Email: devon@petiocol.as
Github: github.com/x