Spinning your Drones with Cadence® Workflows, Apache Kafka®, and TensorFlow

Community Over Code Halifax 2023
Geospatial Track

Paul Brebner
Open Source Technology Evangelist
Instaclustr (by Spot by NetApp)
www.instaclustr.com/paul-brebner/
Simulates, integrates, generates and learns using spatiotemporal data

1. (Simulated) Drones (fly around in space and time)
2. Cadence + Kafka = integration across different Timescales (Slow and Fast)
3. Drone simulation generates lots of spatiotemporal data
4. Incremental Machine Learning over spatiotemporal drifting data is tricky
5. We live in a 4 (at least) dimensional spacetime universe 😊

“Henceforth space by itself and time by itself are doomed to fade away into mere shadows, and only a kind of union of the two will preserve an independent reality” – Minkowski, 1908
Cloud Platform for Big Data Open Source Technologies

A recent addition is Workflow Orchestration with Uber’s Open Source
This Talk Is About Cadence, Kafka, and TensorFlow
Workflow Orchestration (Simple Concept): Task Ordering
Workflow Orchestration (Simple Concept): Task Ordering
Workflow Orchestration (Simple Concept): Task Ordering
Workflow Orchestration (Simple Concept): Task Ordering

But Harder In Practice…
Pedalling with a high Cadence (pedal revolutions) is called **Spinning**!
Mashing or grinding is slow (and bad).

(Source: Shutterstock)
Cadence is Horizontally Scalable—the number of workflows is unlimited (Source: Shutterstock)
Cadence is **Fault-Tolerant**—Workflows can fail!

(Source: Shutterstock)
...And recover (Source: Shutterstock)
Cadence supports Long Running & Scheduled Workflows

(Source: Photo by Soly Moses from Pexels)
@Override
public void startWorkflow(String a) {
    String b = activities.task1(a);
    String c = activities.task2(b);
}
Cadence Architecture
Instaclustr Managed Service

Customer Managed Clients and Workers

Workers
Run workflow logic

Java and Go Clients are supported

Instaclustr Managed Cadence and Database Clusters

Cassandra
PostgreSQL
Kafka
OpenSearch (optional)

 Cadence

Cadence Servers

APIs
How Does Cadence Fault-Tolerance Work?
Event-Sourcing = State Change History + Replaying

State Change History Written to Database –
Write efficient – classic time vs. space (network) tradeoff
Workflow State Recovery by Replaying History

Failure causes complete history replay
Workflow State Recovery by Replaying History

Workflow state and restart point – Read *inefficient* but hopefully infrequent

\[ S_2 = S_0 + \delta_1 + \delta_2 \]
Sleep/Schedule use the same mechanism

Start → Task 1 → Sleep… → Task 2 → End

Database

(Source: Getty Images)
Sleep/Schedule use the same mechanism

Workflow state is recreated when tasks are scheduled to start

\[ S_2 = S_0 + \delta_1 + \delta_2 \]

(Source: Getty Images)
Cadence Activities implement Tasks

Activities (including backwards cycling) can fail

(Source: Shutterstock)
Cadence Activities

- Activities are core to Cadence, they implement tasks
- Remote calls can fail, so wrap them in Activities
- They can contain any code
- Activities are executed at most once and can be automatically retried on failure
- But must be deterministic

![Diagram showing the flow of activities from Start to End with a Remote Call failure]

© Instaclustr Pty Limited, 2023
Good Use Cases?

- 100s to millions of running workflows
- Long running workflows, sleeping and scheduled tasks
- Stateful fault-tolerant applications
- Complex workflows
- Integration with unreliable external systems
- Integration with streaming and event-based systems (e.g. Kafka)
- Short workflows
  - Short workflows (100s of steps)
  - Long workflows (1000s of steps) may take longer to replay history

For example? Financial, retail, delivery!
Spinning your Drones!
A Drone Delivery Application

(Source: Shutterstock)
Drone Workflow Steps

- Charged ready to go at base
- Get an Order to deliver
- Fly to Order location and collect Order
- Fly to Delivery location and drop Order
- Fly back to base
- Recharge
- Repeat
Example (Pirates) of Drone Delivery Flight

3 Legs: Base to Order to Delivery to Base

(Source: Shutterstock)
Drone Way Point Flight Calculations

Returning to Base Example

- Drone flight path is computed in an activity
- Using location, distance, bearing, speed, and charge
- Every 10 seconds
- On failure, the drone won’t crash and will continue flying from the last location

(Source: Shutterstock)
The System: Swim-Lane Diagrams

1st Workflow: Drone Workflow

Start, Ready, Wait for Order, Movement Activities, Recharge, Repeat
Orders Are Also Stateful → Workflow

2nd Workflow: Order Workflow

Start, Generate locations, ready for drone, update locations, End if delivered
Orders Are Also Stateful → Workflow

2nd Workflow: Order Workflow

So, we need coordination between the workflows:

(1) Asynchronous Signaling between workflows

(Source: Shutterstock)
Next, Cadence Meets Kafka

(Source: https://bahumbug.wordpress.com/2014/08/03/kafka-onna-bike/)
Integration With Kafka Adds

(1) Starting a workflow from Kafka
Integration With Kafka Adds

(2) Using a Kafka microservice to coordinate Drone and Order workflows
Step 1: Customer Places an Order—Order Sent to Kafka New Orders Topic
Step 2: Start New Order Workflow—
Kafka Consumer Gets Order, Starts New Order Workflow Using Cadence Client

(Source: Shutterstock)
Step 3: Order Ready for Drone Pickup—
Activity: Send Order Ready Message to Orders Ready Topic
Step 4: Drone Workflow—
Activity: Send Drone Ready Message to Drone Ready Topic, Kafka Consumer
Gets an Order ID and Sends Signal Back to Drone to start the order pickup

(source: Shutterstock)
Step 5: Drone Workflow—Fly To Order

Activity: Fly to Order Location
Step 5: Drone Workflow—Fly To Order And Pickup Order
Step 6: Drone Workflow—Fly To Delivery Location

Activity: Fly to Delivery Location

(Source: Shutterstock)
Step 6: Drone Workflow—Fly To Delivery Location
And Drop Order; Send Location Updates to Order Workflow Every 10s
Step 7: Drone Workflow—Fly To Base

Activity: Fly to Delivery Location; recharge when back at base and start new drone workflow

(Source: Shutterstock)
Step 8: Order Workflow—Update Location

Receive State and Location Signal From Drone Workflow, Update State; If Delivered Then End Workflow
Uber’s Cadence + Apache Kafka = Complementary Timescales

History of the Universe

BICEP2 Collaboration/CERN/NASA

(Source: Wikipedia)
# Uber’s Cadence + Apache Kafka = Complementary Timescales

<table>
<thead>
<tr>
<th></th>
<th>Cadence (Slow Workflows)</th>
<th>Kafka (Fast Streaming Events)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synchronous Events</td>
<td>Asynchronous Events</td>
<td></td>
</tr>
<tr>
<td>Stateful flows</td>
<td>Stateless events</td>
<td></td>
</tr>
<tr>
<td>Sequences</td>
<td>One-off events</td>
<td></td>
</tr>
<tr>
<td>Slow/long-running flows</td>
<td>Fast/instantaneous events</td>
<td></td>
</tr>
<tr>
<td>Sleep/Schedule events</td>
<td>Real-time processing of events</td>
<td></td>
</tr>
<tr>
<td>Complex flow logic</td>
<td>Complex Stream Processing (Kafka Streams)</td>
<td></td>
</tr>
</tbody>
</table>
How Many Drones Can We Fly?

(Source: Shutterstock)
Cluster Details (VCPUS):
Client (8), Cadence (6), Cassandra (18)
Load Test:
2,000 Drones + 2,000 Orders = 4,000 Workflows
20 Drones Flying

Purple = base
Black = drone
Orange = shop
Red = delivery location
Green = successful delivery
Part 2: Kafka+ML (TensorFlow)
over spatiotemporal drifting streaming data

Goal: Predict which shops will be busy

(Source: Getty Images)
Part 2: Kafka+ML (TensorFlow) over spatiotemporal drifting streaming data

Goal: Predict which shops will be busy, or not busy

(Source: Getty Images)
Real-time ML is Everywhere—TikTok+Cat Videos etc.
Incremental Learning Challenges

**Big Data**—lots of data

**Fast Data**—streaming, new data constantly arriving

**Changing Data**—concept drift

**Just-in-time learning**—continuous learning, model must be up to date

Streaming data is infinite, so can’t assume access to all past data

(Source: Shutterstock)
The Drone Learning Problem

The Drone Delivery System

Generates massive amounts of spatiotemporal data

Drone order data sent to Kafka

Can we learn how to predict which shops will be busy in an hour?
ML Architecture

Kafka Streams computes aggregated hourly shop & order details → Busy/NotBusy categorization

Simulation produces lots of streaming spatiotemporal data (drone and order state and locations)

Sent to TensorFlow
“C” Is for Cat!
Mark 1 Perceptron Circa 1960

TensorFlow is a modern neural network framework which supports incremental learning.
Incremental ML—Incrementally!
Experiment 1: “Still” Batch Data Only, No Drift

6,000 samples over a week

Class: Shop busy/not busy per hour
Class and features are computed from hourly aggregated delivery/shop data

Other features include weekday, shop type, average delivery time, etc.

Time is quantized to hours, location to “suburb” (this is “cheating” and ignores actual time/space scale granularity)

Rule is a combination of shop type, location, weekday and hour

(Source: Shutterstock)
TensorFlow Steps

1. Setup Python TensorFlow on your computer
2. Import Pandas and "Numpies"
3. Define data columns (and class label)
4. Read the data into a Pandas DataFrame
5. Split the data into training/evaluation subsets
6. Create a model (design, build, compile)
7. Prepare training data (features and class)
8. Train/fit the model (auto epochs)
9. Evaluate the model
10. Use the model on new data

In step 8, every time the "fit" method is called it *updates* the model – this is critical for Incremental learning.

In step 6, we used the "Adam" optimizer.
Evaluation: What Matters?
Accuracy

Can be misleading unless roughly equal positive/negative samples

(Source: Shutterstock)
Initial Results: 1 week of drone delivery data with shop busy/not busy 50:50 split

Training: Accuracy improved during training from 0.5 to 0.88 over 125 Epochs
Evaluation confusion matrix – also good results
But! Results not repeatable, accuracy (0.5-0.9) and training time (5-92s) vary
Sensitive to batch size (number of samples used per epoch: 8-64, 32 most consistent)
And it sometimes gets stuck in a bad local minima.

<table>
<thead>
<tr>
<th>Actual Class</th>
<th>Predicted Class</th>
<th>TruePositives</th>
<th>FalseNegatives</th>
<th>FalsePositives</th>
<th>TrueNegatives</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>TRUE</td>
<td>1064</td>
<td>76</td>
<td>179</td>
<td>921</td>
</tr>
<tr>
<td>FALSE</td>
<td>FALSE</td>
<td>76</td>
<td>921</td>
<td>179</td>
<td>1064</td>
</tr>
</tbody>
</table>

Data size = 6,000 samples
Shop busy/not busy hourly over a week
Training/evaluation only use a subset
Experiment 2: Incremental Learning—No Kafka, No Drift

**Goals:**
Continuous model training  
From streaming data  
Incremental learning—don’t need all the data  
Model is up-to-date all the time  
Useful for real-time use cases  
Can cope with concept drift in the data (eventually)

**This experiment:**
Same data as last time  
But incremental learning

Incremental clicker games can be addictive  
(Source: Shutterstock)
Approach and Results

**Approach**
- Slicing the data to get 100 samples at a time (simulating Kafka poll)
- Fit with each sample
- Batch 8, 20 fixed epochs
- Evaluate against all previous data

**Results**
- Accuracy improves with time - good
- Max accuracy comparable with Batch learning
- But Oscillation! What’s going on?
Experiment 3: Incremental Learning Using Kafka

“Franz Kafka reading data in the style of Dali”
(Source: Paul Brebner. Dalle-2)
Results:
Better, Not as Much Oscillation

But accuracy worse: 0.72 c.f. 0.9 for batch on same data

X-axis is just “loop” number
Experiment 4: Incremental Learning, Kafka & Concept Drift

(Source: Shutterstock)
Concept Drift at Start of Week 2

Best accuracy during week 1 = 0.7

Concept drift introduced at start of week 2

Accuracy instantly drops to worse (0.3) than guessing (0.5)

Accuracy gradually improves to 0.75

Oscillation still
Experiment 4b: Reset the Model

(Source: Getty Images)
Reset the Model After Drift

If accuracy drops too much, reset the model. Retrain using only new data – removes pre-drift model “inertia”. Helps immediately (accuracy 0.9) But oscillation, and accuracy drops to 0.7 by end of week 2.
Why Oscillation?

We had accidental drift!

Even though the rules weren’t changing, the data was changing over time.

Why? Because temporal features (day of week and time of day) were used to determine shops busy/not busy.

And some hours had no busy shops (end of day!) – so accuracy actually increased.

Accidental Drift
(Source: Shutterstock)
I’m late! I’m late!

Incremental learning may result in a temporal bias due to fixation on recent samples

(Similar to spatial bias from nearby samples?)

(Source: Wikimedia)
Model overfitting (indicated by Oscillation) on time-series data is a well-known problem. Time-series data + incremental learning = down the rabbit-hole (somewhere strange and hard to understand!)
Other (Spatio)Temporal Challenges: Time Encoding (Wrap Around)

Default time encodings have “temporal discontinuities”

E.g. Time wraps around at midnight/midday (12:00 = 00:00)

C.f. Prime Meridian and Anti-Meridian

International date line – not a straight line!

Both temporal and spatial – on West side it’s Today, on the East side it’s Yesterday!
Other (Spatio)Temporal Challenges: Periodicity (Repetition)

Periodicity is common in temporal (and spatial?) data

Ocean Waves have Temporal and Spatial Periodicity

Surfers, Gold Coast, Australia (Source: AdobeStock)
Can We Fix It? Get Rid of Time
But not space

Time to try “no time”
(Source: Shutterstock)
Experiment 4c: No Time Results

No wild oscillations

With reset model, big drop in accuracy initially, but slightly better accuracy by end of week 2.

In production could continue using old model until new one is trained up again.
Conclusions

• Incremental learning is fast, it should be able to keep up with streaming data in a production system

• And incremental learning is able to keep the model up to date (more or less), in conjunction with resetting

• Model accuracy may oscillate due to the presence of time features, fixation on recent data, changes in sample class distribution over time, and concept drift

• Overfitting on timeseries data is something to watch out for and take action to prevent/mitigate - maybe try “Rolling/Sliding Windows” (better for concept drifts)

• Try multiple competing models concurrently

• Potential to use Cadence Workflows for MLOps
Geospatial Highlights

- Simulated spatiotemporal Drone Delivery system
  - Cadence + Kafka integrated different timescales (slow and fast)
  - Cadence Scheduled Activities for movement calculations

- Generated lots of spatiotemporal data

- ML over spatiotemporal data

- Spatiotemporal challenges
  - Detecting and relearning temporal drift is tricky – similar problem for spatial drift?
  - Simplified precomputed aggregated spatiotemporal quantization (hour, suburb)
  - Watch out for
    - spatiotemporal encoding discontinuities, wraparounds and periodicity
    - bias and overfitting in incremental learning due to focus on the now (and here for spatial data)
Spatial/Temporal Bias Is Inevitable

Observations are always from a single point in space and time—you only directly experience “here” & “now”, and can only inspect past events (not future events), and space is “big”!