Teralytics unlocks powerful consumer insights from telecom data.

We work with some of the largest telecom operators in North America, Europe and Asia as their trusted data monetization partner.

**About Us**

**Device Movement & Consumer Intelligence Data**
Rich behavioral data combined with accurate mobile location data – on a massive scale

**Cutting Edge Technology**
We have perfected the ingestion, modeling and transformation of raw signals from user actions and movements on operator networks to deliver meaningful and actionable insights. We are the brainchild of two graduate students and a professor in ETH Zürich.

**Solid Backing**
$40M in funding from co-founder of Skype and early investors in Facebook, Spotify, and Airbnb. 60+ employees; 30 top PhDs and data scientists
Outline

- Project background and tech
- Intro to Akka Streams
- Stream processing under failure
- Real world use case
What have we built?

- Low latency (near real-time).
- Scalability from 10k to 100k events per second.
- Exactly-once stateful processing.
- High availability and no delays under failure.
- Low maintenance (self recovering).
- Minimal footprint in terms of CPU and RAM.
- Kafka, Scala, Akka tech stack.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stream processing</td>
<td></td>
</tr>
<tr>
<td>Highly available</td>
<td></td>
</tr>
<tr>
<td>Lightweight</td>
<td></td>
</tr>
</tbody>
</table>
Transportation use case

Near real-time view of train/subway operations

- Supplements existing systems.

For operators, regulators, and planners

- Optimize operations.
- Emergency management.
- Event preparedness.
- Mobility demand planning.
Stream processing
High-level overview

Cellular network events → Dataflow stream processing → Analytics results
Train detection data flow

Cellular network events

Ingestion → Train Trip Detection → Platform Crowd Calculation

Station Transition Detection → Train Info Calculation

Analytics results
Intro to Akka Streams
Akka Streams

- Allows you build arbitrary stream processing topologies/graphs.
- Built on top of Akka actors.
- Distributed within a single JVM.
- Full flow-control/backpressure.
Concepts

- **source**
- **sink**
- **flow**

- **fan in**
- **fan out**
Composition

source

fan in

flow

source

fan in

flow

flow

flow

flow

fan out

flow

sink

flow

TERALYTICS
Runnable graph
Stream processing under failure
Train detection data flow

- Cellular network events

- Ingestion

- Train Trip Detection

- Platform Crowd Calculation

- Station Transition Detection

- Train Info Calculation

- Analytics results

Local state
Asynchronous Barrier Snapshotting (ABS)

- Implemented for Apache Flink.
- Based on inserting markers into the execution graph.
- Markers flow with the stream and trigger state snapshotting in each processor.
- Snapshotting is asynchronous and does not require the system to halt.
Insert barrier marker
Trigger state snapshotting
Trigger state snapshotting
Trigger state snapshotting
Wait for the marker on input channels
Wait for the marker on input channels
Complete the snapshot
Trip detection use case
Train detection dataflow

Ingestion → Train Trip Detection → Platform Crowd Calculation → Analytics results

Cellular network events → Station Transition Detection → Train Info Calculation
Problems to solve

- Fault tolerance.
- Scalability.
- High availability.
Fault tolerance

- Checkpoint processing state periodically (ABS).
- Recover state as fast as possible in case of failure.
ABS state snapshotting
State snapshotting
State snapshotting
State snapshotting

- Ingestion
- Marker Inserter
- Snapshot Committer
- Train Trip Detection
- Station Transition Detection
- Train Info Calculation
- Platform Crowd Calculation
State snapshotting
State snapshotting

Ingestion

Train Trip Detection

Platform Crowd Calculation

Station Transition Detection

Train Info Calculation

Marker Inserter

Snapshot Committer
State snapshotting

Ingestion -> Train Trip Detection -> Platform Crowd Calculation

Marker Inserter

Snapshot Committer

Commit
High availability with zero delay under failure
Event-time windowing


Windowing data in Akka Streams

Akka Streams provide a lot of combinators to manipulate data streams out-of-the-box, but one missing piece of functionality that I needed recently is the ability to window data based on time. In stream processing systems this is quite a well known operation, we can find it e.g. in Flink, Kafka streams or Spark; however in Akka Streams we have to use some
Recap

- Lightweight, Highly-available Stream processing.
- Recovery using ABS.
- Scalability, fault tolerance, HA in the real world.
Takeaways

- Kafka and Akka Streams already provide the basics.
- Akka Streams can be extended to HA, exactly-once processing.
- Full frameworks are not always needed.
Thank you.