




# Time/Space Sessions with Apache Beam

How to extend Apache Beam to produce location-aware sessions

Davide Anastasia

LinkedIn: <https://uk.linkedin.com/in/davideanastasia>

Twitter: <https://twitter.com/davideanastasia>

- 
- GitHub: <https://github.com/davideanastasia/apache-beam-spatial>
    - Source code for all the algorithms described
    - Source code for the demo
    - Presentation in PDF format
    - Instructions on how to run the demo in your GCP environment
  - Time/Space Sessions in Apache Beam (Part 1):  
<https://medium.com/@davide.anastasia/time-space-sessions-in-apache-beam-b402cdf8470>



# Apache Beam

Apache Beam is a library that unifies batch and streaming data processing into the same programming model

Leverages a runner to perform the actual task (Apache Flink, Google Dataflow, Apache Spark and many others)

Specifically designed to incorporate the concept of time in the data processing graph.

One of the few to incorporate both processing time and event time in the framework



# Apache Beam: Windowing

Three different windowing strategies:

- Fixed window
  - Example: every minute, count the completed transactions
- Sliding window
  - Example: every minute, sum the last 5 minutes of completed transactions
- Session



# Session

Sequence of events terminated by a gap of inactivity greater than some timeout [[1](#)]

Not predefined length (i.e. data driven)

Unaligned windows

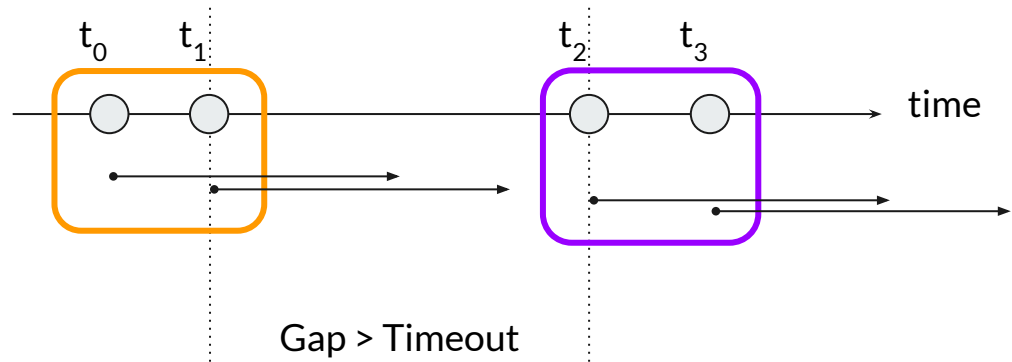
Useful for:

- Summarization
  - Smaller disk usage
  - Faster analytics
  - Example: Calculation of dwell time

# Session Algorithm

Sessions are created sorting the items and merging them when the time interval between elements is lower than a predefined timeout

If the interval between elements is bigger than the timeout, no merge is performed (new session is created)





# Location Data: Challenges

Location data is continuous over two dimensions (can be tricky to process)

Using the standard algorithm will create sessions with no spatial awareness

Solution:

- Split space into “tiles”, using well-known algorithms like Uber H3, Google S3 or Geohash (convert latitude/longitude to an *address*)
- Use tile ID in the sessionization algorithm to decide whether merge or not

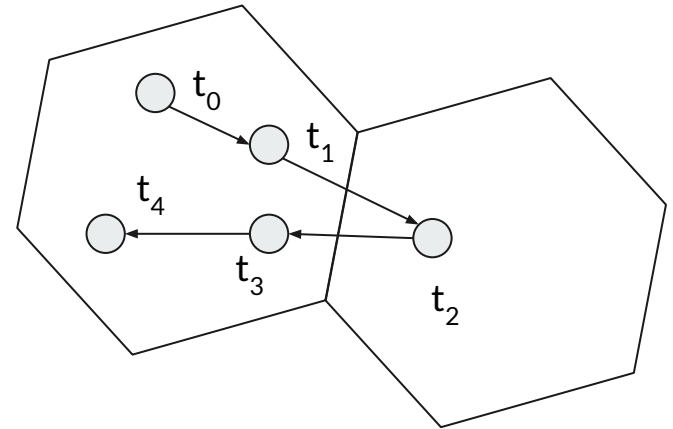
# Location + Session Algorithm

Two approaches (different sorting strategy before merge):

- Time Only
- Space & Time

Regardless of the strategy, a new session is created when either

$t_{n+1} - t_n > \text{threshold}$   
or when the tile ID is different

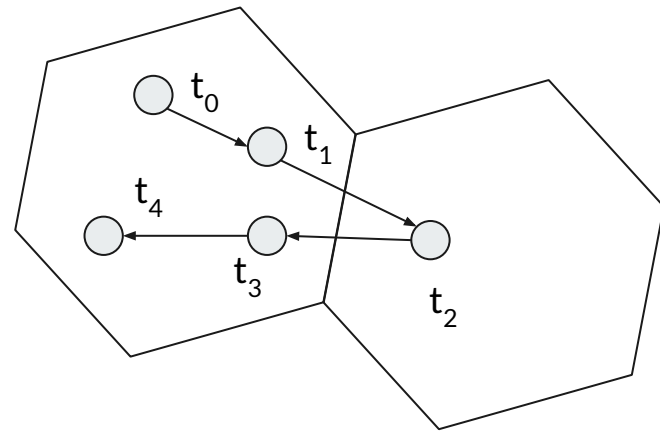
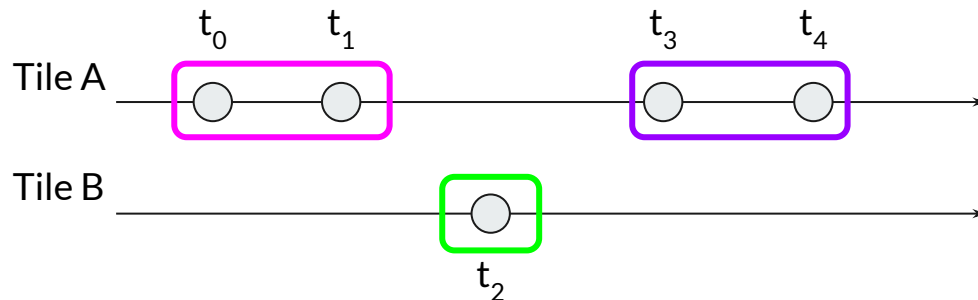




# Time Only Strategy

Works well when data is reasonable clean

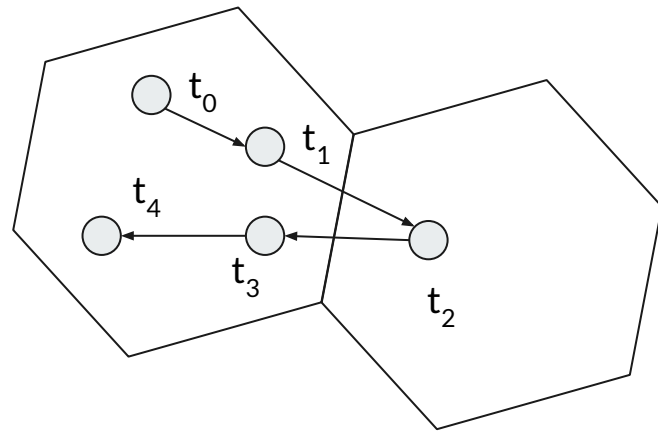
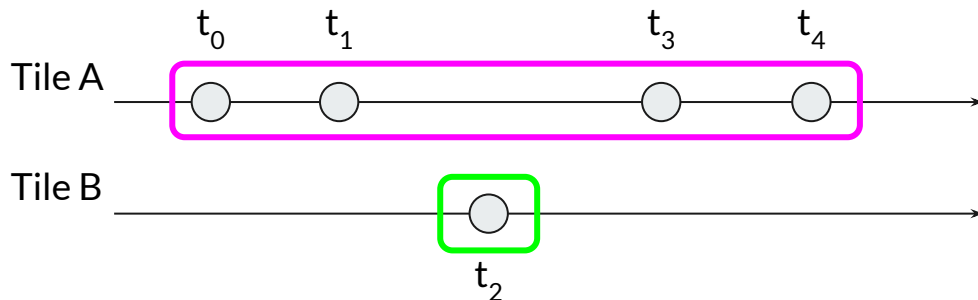
Produces non-overlapping sessions



# Space & Time Strategy

Works better in the presence of many outliers

Produces overlapping sessions





# Use cases

Some potential uses cases are:

- Additional feature in machine learning models (transform latitude/longitude in a categorical feature) [[2](#), [3](#)]
- Advertising (location-based targeting), Marketing (where are my users coming from?), Forecasting (what's the average time spent in a location?)
- Realtime tracking of vehicle/goods

...and many more!

---

# Demo



# Demo

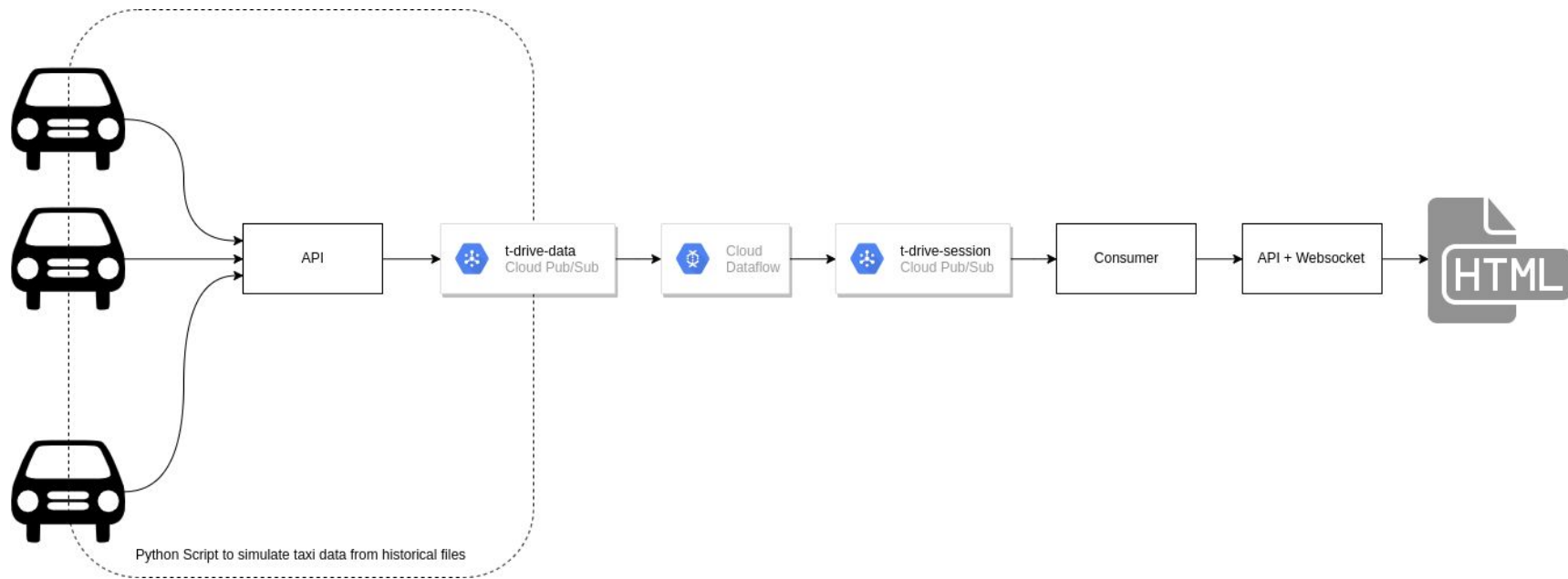
Google Dataflow, Google Pubsub,  
Flask, Websocket, Leaflet.js

Reference dataset: [Microsoft T-Drive](#) (one week of taxi location data in Beijing)

Taxi samples are published on a Cloud Pub/Sub topic

Samples consumed by a Google Dataflow job running an Apache Beam pipeline, producing sessions

Sessions pushed through a Websocket into a Leaflet.js map



# Thanks!

## Any question?

Davide Anastasia

LinkedIn: <https://uk.linkedin.com/in/davideanastasia>

Twitter: <https://twitter.com/davideanastasia>

