Geospatial and Temporal Forecasting at Uber

September 09, 2019
Apachecon

Chong Sun, Brian Tang
01 Marketplace Forecasting at Uber
02 Geospatial Representation
03 Geospatial Processing
04 Use Cases
Marketplace Forecasting at Uber

What is marketplace and how does forecasting fit in?
Marketplace Forecasting

Real Time Forecasting
Minute-level forecasts 1-2 hours into the future

Near-Term Forecasting
10-15 minute-level forecasts several hours into the future

Long Term Forecasting
Hour level forecasts 1-2 weeks into the future
Estimated Time to Request

Help drivers decide if they should wait

Use historical and recent signals to predict the future wait times across cities and airports.
Suggestions

Help drivers decide if they should move

Suggest locations with more opportunities for matching with riders and help them navigate there.
Geospatial Representation

How does Uber see the world?
Hexagons!
Partition the world
Run algorithms
Hexagonal Hierarchical Geospatial Indexing System

H3 is an open source geospatial indexing system using a hexagonal grid that can be (approximately) subdivided into finer and finer hexagonal grids, combining the benefits of a hexagonal grid with S2’s hierarchical subdivisions.
H3 Resolutions

Hierarchical subdivisions
With the largest resolution roughly the size of continents down to the smallest resolution of a meter squared. The library gives flexibility in the size of hexagon to work with.
Why hexagons?
Why hexagons?

Uniform adjacency
Hexagons have no ambiguous neighbors

Low shape and area distortion
Hexagons can fill an icosahedron and offer low distortion
Global Grid

Few tradeoffs
- Not completely uniform shape
- Not perfect child containment

Many advantages
- Uniform edge length
- Uniform angles
- Optimally compact
- Optimally space-filling
- Uniform adjacency
- Hierarchical relationships
- Low shape distortion
- Low area distortion
Geospatial Processing

Working with hexagons
Hexagon Data

Uber on Hexagons
Many decisions are made on small hexagons

Hexagons Level 9
Larger cities have 500k+ hexagons

Sparse Data
E.g., a few requests in some hexagons for a whole day
Hexagon Data Smoothing

**Hexcluster**
Clustering hexagons into groups and use the aggregated values of all the hexagons in each cluster
- Low computation
- “Arbitrary” boundaries

**Kring smoothing**
For each hexagon, using the aggregated values of all its k ring neighbours
- Heavy computation
- Flexible
Hexagon Data Smoothing

**Hexcluster**
Clustering hexagons into groups and use the aggregated values of all the hexagons in each cluster
- Low computation
- "Arbitrary" boundaries

**Kring smoothing**
For each hexagon, using the aggregated values of all its k ring neighbours
- Heavy computation
- Flexible

<table>
<thead>
<tr>
<th>Ring</th>
<th># hexagons</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-ring</td>
<td>6</td>
</tr>
<tr>
<td>2-ring</td>
<td>12</td>
</tr>
<tr>
<td>3-ring</td>
<td>18</td>
</tr>
<tr>
<td>k-ring</td>
<td>6 * k</td>
</tr>
</tbody>
</table>

M hexagons K-ring data smoothing computation:

\[ M \times K \times \left( 6 + K \times 6 \right) / 2 \]
Convolution

(2-D) Convolution
Slide a kernel (small matrix) on top of an input (big matrix), multiple and add the corresponding values to produce the convolution result.

A base component of CNN (Convolutional Neural Network) in deep learning.

Efficient Implementation
Many efficient implementations of convolution in popular packages, e.g., Scipy, TensorFlow, PyTorch.

GPU Acceleration
GPU is a perfect fit for accelerating convolution computation.
Convolution

(2-D) Convolution

Slide a kernel (small matrix) on top of an input (big matrix), multiple and add the corresponding values to produce the convolution result.

A base component of CNN (Convolutional Neural Network) in deep learning.

Efficient Implementation

Many efficient implementations of convolution in popular packages, e.g., Scipy, TensorFlow, PyTorch.

GPU Acceleration

GPU is a perfect fit for accelerating convolution computation.
Convolution

(2-D) Convolution
Slide a kernel (small matrix) on top of an input (big matrix), multiply and add the corresponding values to produce the convolution result.
A base component of CNN (Convolutional Neural Network) in deep learning.

Efficient Implementation
Many efficient implementations of convolution in popular packages, e.g., Scipy, TensorFlow, PyTorch.

GPU Acceleration
GPU is a perfect fit for accelerating convolution computation.
Hexagon Convolution

Hex Convolution
Conceptually, convolution on hex is similar to convolution on square grid matrix

Filter
Using different weights in the filter generates different convolution results. E.g., weighted sum.
Kring data smoothing could be done by using convolution with weight 1 for each hexagon of K rings, e.g, 1-ring smoothing

Challenge
None of known convolution implementations is for hexagon coordinate systems
Optimization and GPU acceleration could not be applied to hexagons directly
### Hexagon Convolution

**Hex Convolution**
Conceptually, convolution on hex is similar to convolution on square grid matrix.

**Filter**
Using different weights in the filter generates different convolution results. E.g., weighted sum.

Kring data smoothing could be done by using convolution with weight 1 for each hexagon of K rings, e.g, 1-ring smoothing.

**Challenge**
None of known convolution implementations is for hexagon coordinate systems.

Optimization and GPU acceleration could not be applied to hexagons directly.
Hexagon Convolution

Hex Convolution

Conceptually, convolution on hex is similar to convolution on square grid matrix.

Filter

Using different weights in the filter generates different convolution results. E.g., weighted sum.

Kring data smoothing could be done by using convolution with weight 1 for each hexagon of K rings, e.g, 1-ring smoothing.

Challenge

None of known convolution implementations is for hexagon coordinate systems.

Optimization and GPU acceleration could not be applied to hexagons directly.
Hexagon Coordinate System (1)

Cube Coordinate

- Represents 2-D hexagon grid in a 3-D Cube.
- Memory inefficient
- No direct map from cube coordinate to a 2-D rectangular grid
Hexagon Coordinate System (2)

Double Coordinate
- Two double coordinates
  - Double coordinates by heights (example)
  - Double coordinates by widths
- Easy map to square grid based on the coordinate values
- Very inefficiency for convolution as the cells are not contiguous
Hexagon Coordinate System (2)

Double Coordinate
- Two double coordinates
  - Double coordinates by heights (example)
  - Double coordinates by widths
- Easy map to square grid based on the coordinate values
- Very inefficiency for convolution as the cells are not contiguous
Hexagon Coordinate System (3)

Axial Coordinate

- Map to the square grid well with missing cells in the corner
- Good fit for the convolution with extra filtering
Hexagon Coordinate System (3)

Axial Coordinate

- Map to the square grid well with missing cells in the corner
- Good fit for the convolution with extra filtering
Hexagon Convolution (0)
Hexagon Convolution (1)
Hexagon Convolution (2)

Input

Filter

\[
\begin{array}{ccc}
1 & 2 & 2 \\
1 & 2 & 3 \\
0 & 0 & 4 \\
2 & 1 & 6 \\
1 & 4 & 5 \\
\end{array}
\quad \star 
\begin{array}{ccc}
1 & 1 \\
1 & 1 \\
1 & 1 \\
\end{array}

\Rightarrow
\begin{array}{ccc}
9 & 16 \\
6 & 16 & 30 \\
13 & 29 \\
\end{array}
\]
Hexagon Convolution (3)
Kring Data Smoothing

Kring Data Smoothing
Smoothing hexagon data with values in kring the neighbour hexagons.

- Weights for kring smoothing
  - Equal weight kring smoothing
  - Dynamic weight kring smoothing
- Dynamic kring size for smoothing
  - Run the smoothing for all the kring sizes separately
**Kring Data Smoothing**

Smoothing hexagon data with values in kring the neighbour hexagons.

- **Weights for kring smoothing**
  - Equal weight kring smoothing
  - Dynamic weight kring smoothing

- **Dynamic kring size for smoothing**
  - Run the smoothing for all the kring sizes separately
Kring Smoothing Performance

Performance Comparison

- Basic implementation
- Convolution approach with CPU
- Convolution approach with GPU

100x faster than k-ring smoothing with GPU and 10X faster with CPU
Use cases

How is this used in production?
Every minute

30M+  700+  10+

Hexagons smoothed and forecasted
Cities worldwide
Quantities forecasted
High throughput and low latency
Developed as a true streaming framework from the ground up, Flink has enabled us to produce forecasts with only a few seconds of latency.

Efficient memory management
Even with complex custom aggregations we have been able to keep memory utilization under 60%.

Strong feature set
Operator isolation with keyby lets us use city specific configurations and the Table API offers really nice high level abstractions.
Significant savings

The combination of both moving our pipelines to Flink and leveraging hexagon convolution has reduced the total required core count by ~90% and memory utilization per box by ~70%
Forecasting Flink Pipeline
Summary

- Uber data is aggregated on hexagons using the H3 library.
- Geospatial processing is expensive. Leveraging convolution allows for significant performance gains.
- Geospatial processing is expensive. Efficient pipeline frameworks are critical with Flink being a very natural fit.
Q&A

Additional resources:

- Uber Marketplace
- Uber github