FEATURE STORE SUMMIT

12-13 OCTOBER | 08:30 AM - 4:00 PM PT

ORGANIZED BY HOPSWORKS
Michelangelo Palette at Scale

Amit Nene
Architect, Manager
Uber

Nicholas Marcott
Lead Engineer
Uber
Michelangelo Palette

Feature Preparation
Batch + Streaming ETLs

Monitoring
Pipeline & Data Quality

Palette Feature Store
Online serving + Joins
Offline serving + Joins

Feature Discovery
Sharing across Models
Automatic feature selection

Michelangelo Transformer
Model specific feature xforms
History

Self-Service ETL and automation
Online Serving

World’s first Feature Store announced!
Early Adoption

Scalability
Dispersals
Optimized Serving

Near real-time features
Uber-wide adoption

Automatic Feature Selection
Data Quality

Problems at Scale

Online Serving
10s of Millions of QPS

Offline Serving
100s of TBs per training run

Pipeline Scale
1000s of Feature Pipelines

Discovery at Scale
Select from 100K features
Online Serving

- Restaurants: x 1000s
- Features: x 10s
- Models: Final ranking

Uber Engineering
Online Serving

Serving Infra

Table Consolidation

Custom Partitioning

Efficient Dispersals
Serving Infra

**Local cache**
- JVM-local, in-memory
- Thousands of cities (100s of MBs)
- Hottest partitions

**Remote cache**
- Distributed, in-memory (eg. Redis)
- Millions of restaurants (100s of GBs)
- Hot partitions

**NoSQL**
- Distributed, KV-Store (eg. Cassandra)
- 100 Millions of Users (10s of TBs)

QPS/Latency per $ → Storage Capacity

Advanced uses
Table Consolidation

Feature Group Table Design
- Ownership
- Feature Gen Job
- Semantic Grouping

<table>
<thead>
<tr>
<th>Restaurant (Key)</th>
<th>Ratings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Starbux</td>
<td>4.7</td>
</tr>
<tr>
<td>Fills</td>
<td>4.8</td>
</tr>
<tr>
<td>Black Bottle</td>
<td>4.9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Restaurant (Key)</th>
<th>Wait Times</th>
</tr>
</thead>
<tbody>
<tr>
<td>Starbux</td>
<td>5min</td>
</tr>
<tr>
<td>Fills</td>
<td>10min</td>
</tr>
<tr>
<td>Black Bottle</td>
<td>20min</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Restaurant (Key)</th>
<th>Embeddings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Starbux</td>
<td>[0.4, 0.1, ...]</td>
</tr>
<tr>
<td>Fills</td>
<td>[0.9, 0.2, ...]</td>
</tr>
<tr>
<td>Black Bottle</td>
<td>[0.5, 0.4, ...]</td>
</tr>
</tbody>
</table>
Table Consolidation

Query Fanout Per Table

High QPS

High Tail Latency

Feature Store

Restaurant | Ratings
---|---
Black Bottle | 4.9

Restaurant | Wait Times
---|---
Black Bottle | 20min

Restaurant | Embeddings
---|---
Black Bottle | [0.5, 0.4, ...]
Table Consolidation

Consolidated Tables

Eliminate fanout

Low Latency

<table>
<thead>
<tr>
<th>Restaurant</th>
<th>Ratings</th>
<th>Wait Times</th>
<th>Embeddings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Bottle</td>
<td>4.9</td>
<td>20min</td>
<td>[0.5, 0.4, ...]</td>
</tr>
</tbody>
</table>

Black Bottle features request

Feature Store

Uber Engineering
Custom Partitioning

Features based on 2 or more keys
Partition key = Primary Key = User + Restaurant

<table>
<thead>
<tr>
<th>User (Key 1)</th>
<th>Restaurant (Key 2)</th>
<th>Avg Rating (Value)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nicholas</td>
<td>Starbux</td>
<td>4.5</td>
</tr>
<tr>
<td>Nicholas</td>
<td>Fills</td>
<td>4.6</td>
</tr>
<tr>
<td>Nicholas</td>
<td>Black Bottle</td>
<td>4.0</td>
</tr>
</tbody>
</table>
Custom Partitioning

Query Fanout Per Restaurant

High QPS

High Tail Latency

Nicholas's ratings request

Feature Store

Nicholas Starbux 4.5

Nicholas Fills 4.6

Nicholas Black Bottle 4.0
Custom Partitioning

Customized partitioning
Partition Key = User
Shard Local Key = Restaurant

Eliminate fanout

Low Latency
Nicholas’s ratings request

<table>
<thead>
<tr>
<th>User</th>
<th>Restaurant</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nicholas</td>
<td>Starbux</td>
<td>4.5</td>
</tr>
<tr>
<td>Nicholas</td>
<td>Fills</td>
<td>4.6</td>
</tr>
<tr>
<td>Nicholas</td>
<td>Black Bottle</td>
<td>4.0</td>
</tr>
</tbody>
</table>
Efficient Dispersals

Too many SSTable files created

Compaction falls behind

High Read Latency
Efficient Dispersals

Cassandra and Spark partition functions aligned

Tuning Compaction algorithms

Consistency, Read repair, GC, SSD config, etc.

Spark

Cassandra SSTables
Offline Serving

How to tune Spark for huge joins?

How to avoid skewed joins and OOMs?

How to speed up joins?
Batching

Break join into manageable batches

Fixed tuning per batch

3 months of Data  \rightarrow  1 month * 3
# Delta

Process updates instead of entire snapshots

<table>
<thead>
<tr>
<th>User (Key)</th>
<th>Day 1</th>
<th>Day 2</th>
<th>Day 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nicholas</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>Amit</td>
<td>21</td>
<td>21</td>
<td>22</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>User (Key)</th>
<th>Day 1</th>
<th>Day 2</th>
<th>Day 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nicholas</td>
<td>10</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Amit</td>
<td>21</td>
<td>-</td>
<td>22</td>
</tr>
</tbody>
</table>
Delta

10X or more reduction in Spark shuffle cost
Other Join Optimizations

Spark optimizations
Job Scheduling, Map-side joins, Filtering, Large Containers

Join reuse framework
Reuse joins across multiple training iterations (e.g. hyperparameter tuning)
Pipelines at scale

Are 1000s of Pipelines are producing reliable features?

How can we avoid debugging of issues at Training time?

How can we enforce accountability?
Uber Data Quality

Metadata

Lineage

Tiering

Test Registration

Ownership
Tying to ML Quality

Feature Store
Auto-onboard for Data Quality

Michelangelo Model Score
Data Quality as score component

Leadership Visibility
Poor scoring models flagged
Discovery at Scale

How to choose relevant features from repository of 100K+?

How do we avoid redundant building of features?
Feature Store Search

Search by entity, key, name, etc.

Databook
Explore Ownership Create

ML Features
• store.

Tier Team
Tier 1 Eats

Definition Quality

Description
HQL Source

Join Key
user_uuid

Features
Name Feature Type

Source Datasets
Offline:
uber_eats
Online:
store

Compute Type
BATCH
Data Browser

Stats, Data quality

Lineage

Uber Engineering
Automatic Feature Search

Entropy of a feature

\[ H(X) = - \sum_{i=1}^{n} p(x_i) \log_2 p(x_i) \]

Mutual Information: Feature and Label

\[ I(X; Y) = H(X) - H(X|Y) \]

Search Feature Store by Join Key, rank by MI

Michelangelo Training workflow

High level workflow of optimal feature discovery
Where we’re headed

Recommendation Systems
Uber Search Engine
Inline execution of Models

Feature Intelligence
Lineage across Data and ML
Data Mining tools

Embeddings
Vector types, Versioning, Discovery

Near real-time features
Aggregation infra
Seamless backfills
Thank you!

Do you have any questions?

https://www.linkedin.com/in/amitabh-nene/

https://www.linkedin.com/in/bmarcott/