“Big Data Processing the UBER Way”
Praveen Murugesan
Uber’s Mission

“Transportation as reliable as running water, everywhere, for everyone”

75+ Countries

500+ Cities

And growing...
Agenda

- UBER’s Data Audience
- Data Infra - A Brief History
- What we Solved
- What we are Currently Solving
Uber’s Data Audience

● 1000s of City Operators (Uber Ops!)
  ○ On the ground team who run and scale uber’s transportation network

● 100s of Data Scientists and Analysts
  ○ Spread across various functional groups including Engineering, Marketing, BizDev etc

● 10s of Engineering Teams
  ○ Focussed on building automated Data Applications
Uber’s Data Audience

● 1000s of City Operators (Uber Ops!)
  ○ driver funnel, rider retention, business performance and other daily/weekly reports

● 100s of Data Scientists and Analysts
  ○ A/B experimentations, Spend analysis etc

● 10s of Engineering Teams
  ○ real-time fraud detection, map search, location prediction etc
Data Infrastructure Today

- Kafka Logs
- Schemaless DB
- SOA DBs
- HDFS
- Spark | Presto
- Hive
- ETL
- Machine Learning
- Experimentation
- Data Science
- City Ops
- Data Science
- Adhoc Analytics
A Things we solved along the way..

- **Scalable Ingestion Model**
  - home-grown streaming ingestion solution
  - [https://eng.uber.com/streamific/](https://eng.uber.com/streamific/)

- **Built a Hadoop Data Lake**
  - No more limited to storage, (EL from Data Sources instead of ETL)
  - JSON -> Avro -> Parquet
A Things we solved along the way..

● **Strict Schema Management**
  ○ Because our largest data audience are SQL Savvy! (1000s of Uber Ops!)
  ○ SQL = Strict Schema

● **BigData Processing Tools Unlocked - Hive, Presto and Spark**
  ○ Migrate SQL savvy users from Vertica to Hive & Presto (1000s of Ops & 100s of data scientists & analysts)
  ○ Spark for more advanced users - 100s of data scientists

● **GeoSpatial Computation Platform**
  ○ Because everyone runs geo based Queries

● **Data Tools**
  ○ Spark UDK - To reduce barrier to entry for writing Spark Jobs
  ○ Attis - A tool to analyze query costs and status
Pre-Strict Schema Management

Data Producers -> Json -> Data Warehouse -> Data Consumers
```
{  
  created_at : "Dec 1, 2015"
}

CREATE (  
  created_at: string
)

parseDate(data.created_at, "%M %d, %yy")
```
Pre-Strict Schema Management

```json
{
    created_at: 1459404179.03
}
```

CREATE (created_at: string)

parseDate(data.created_at, "%M %d, %yy")
Solution: Centralized Schema Management
Solution: Centralized Schema Management

```
{  
  created_at: 1459404179.03  
}
```

```
CREATE (  
  created_at: string  
)
```

```
parseDate(data.created_at, "%M %d, %yy")
```
Solution: Strict Centralized Schema Management

- A central versioned schema Contract for every dataset
  - used by teams, producers and consumers to negotiate data contracts
  - we use Heatpipe - an uber library which is a wrapper over Apache Avro as the serialization format
- A schema evolution system
  - Which ensures schemas evolution is compatible with previous data
  - Strictly typed
- A web UI schema manager
  - To easily create, edit, consume avro schemas.
  - Serves as documentation for data
Avro Schema Example

```json
{
    "namespace": "tulip.marketing_email_events",
    "type": "record",
    "name": "subscriptions",
    "fields": [
        { "name": "user_uuid", "type": "string" },
        { "name": "territory_uuid", "type": { "type": "array", "items": "string" } },
        { "name": "territory_et_fields",
            "type": { "type": "array", "items": { "type": "map", "values": "string" } } }
    ]
}
```
Schema Manager Web Ui aka Watchtower
Schema Evolution Venn

Heatpipe
Avro
Parquet
Hive
Presto
Parquet for Hadoop Data Lake Storage

- **Supports Schema**
- **2 to 4 times faster than json/gzip**
  - column pruning
  - wider nested table support (at uber)
  - filter predicate push-down
  - columnar compression
- **Strong Open Source Support**
  - Hive, Presto, Spark
Queryable Big Data Warehouse (2016)
But Hive is Slow..

Vertica

Fast…
but cannot scale cheaply

Hive

Scales cheaply and reliably...
but is not fast
Queryable Big Data Warehouse (2016)

Hadoop Data Lake

Hive (Batch SQL)  
Presto (Interactive SQL)
Queryable Big Data Warehouse (2016)

JANUS
(ANSI SQL Federation Gateway)

Hive
(Batch SQL)

Presto
(Interactive SQL)

Hadoop Data Lake
Query Federation

- Adhoc/Scheduled SQL
  - Query via Janus - Gateway Service
  - Uses ANSI SQL standard (Presto, Hive underneath today..)
  - Dynamically picks YARN Queues
  - Keep bad queries out!
Query Engine Enhancements

● Presto
  ■ Nested Column Pruning for Parquet Columns
    ● Making Presto fasterr!
  ■ Geospatial support
    ● Filling in the UBER Gap!

● Hive
  ■ Hive on Parquet schema evolution fixes
Attis - Our Query Monitoring Tool

- **Oracle AWR like reports**
  - top queries by CPU
  - top queries by runtime

- **Cost Analysis**
  - Approx Cost to run query on AWS (by CPU and Memory)

- **Realtime Query Tracker**
## Query Engine Monitor

### In Flight Queries

<table>
<thead>
<tr>
<th>Query ID</th>
<th>State</th>
<th>Query Statement Preview</th>
<th>Engine</th>
<th>Queue</th>
<th>Progress Status</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>20160913_191047_04559_fer6k</td>
<td>RUNNING</td>
<td><code>select msg.session_id, msg.rider_app.rider_id, msg.counter, msg.rider_app.trip_id, msg.r...</code></td>
<td>Presto</td>
<td>N/A</td>
<td>100%</td>
<td>$5.025</td>
</tr>
<tr>
<td>20160923_215139_06716_nws88</td>
<td>RUNNING</td>
<td><code>select msg.rider_app.device.os, msg.rider_app.version, ts, from_unixtime(ts) as formatted_ts, datestr, m...</code></td>
<td>Presto</td>
<td>N/A</td>
<td>99%</td>
<td>$2.205</td>
</tr>
</tbody>
</table>

### Finished Queries

<table>
<thead>
<tr>
<th>Query ID</th>
<th>Status</th>
<th>Query Statement Preview</th>
<th>Engine</th>
<th>Start Time</th>
<th>Elapsed Time</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>hive_2016092481438_79de0da0-e8c2-4af-e64-c09d0b973e68</td>
<td>SUCCEEDED</td>
<td><code>SELECT datestr as date, city_id, case when request_device = 'iphone' then 'ios' else request_device ...</code></td>
<td>Hive</td>
<td>Sat Sep 24 2016 11:14:40</td>
<td>0h:4m:57s</td>
<td>$2.52</td>
</tr>
</tbody>
</table>
What next to solve for Data Warehousing?

- **True Query Federation**
  - Predict if a Query should be run on Hive or Presto?

- **Query Translation**
  - Can we convert expensive Presto Queries to Hive

- **UDF Management across Hive/Presto**
SQL solves for the most part....
But, What about Complex Data Applications?

- Machine Learning algos
- Low Latent batch processing
- Stitching HDFS files
- etc
SQL solves for the most part....
But, What about Complex Data Applications?

- Machine Learning algos
- Low Latent batch processing
- Stitching HDFS files
- etc

Use Spark!
Spark UDK (Uber Developer Kit)

Goal:

● **Self-Serve Development kit:**
  ○ Reduce barrier to entry for new spark users
  ○ Application Lifecycle management
    ▪ Scheduling, Monitoring etc

● **Abstract our runtime environment**

● **Ensure a reliable multi-tenant infrastructure**
Spark UDK

<table>
<thead>
<tr>
<th>API</th>
<th>Engineers</th>
<th>SRE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tools</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Spark UDK

- Engineers
  - Easy
  - Self-Service
  - Multi-Platform

- SRE
  - No Harm
  - Reliability
Spark UDK Engineering APIs

● **SCBuilder**
  ○ Encapsulates cluster environment details
  ○ Perf, debug optimized (history, event logs, YARN configs)
  ○ SRE approved CPU & Memory settings

● **Data Dispersal**
  ○ Kafka Dispersal
    ▪ RDD - Parallelization
    ▪ HA, Rate - limiting, schema enforcement
    ■ publish(data: RDD, topic: String, schemaId: Int, appId: String)
  ○ Also have connectors to Hive, Elastic Search
Spark UDK Tools

- **Sparkplug**
  - A collection of popular job templates
  - Two commands to run the first job in Dev
  - One use case per template
    - e.g. Ozzie + SparkSQL + Incremental processing
    - e.g. Incremental processing + Kafka dispersal
  - **Best Practices**
    - built-in unit tests, test coverage, Jenkins
    - built-in Kafka, HDFS mocks
<table>
<thead>
<tr>
<th>Future Work</th>
<th>Engineers</th>
<th>SRE</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- SCBuilder</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Kafka dispersal</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Hive table registration</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Incremental processing</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Geo-spatial processing</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Debug logging</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Metrics</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Configurations</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Data Freshness</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Resource usage</td>
</tr>
<tr>
<td>Tools</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Distributed Debugger</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- SparkPlug</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Unit testing</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Oozie integration</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Resource usage auditing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Data access auditing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Machine learning on jobs</td>
</tr>
</tbody>
</table>
within(trip_location, city_shape)
Find if a car is within a city

contains(geofence, auto_location)
Find all cars in an area
Uber Geospatial Processing

overlaps(trip1, trip2)
Find trips that have similar routes

intersects(trip_location, gas_locations)
Find all gas stations a trip has passed by
Spatial Join: The Problem

**Objective:** Associate all trips with city_id for a single day.

```sql
SELECT trip.trip_id, city.city_id
FROM trip JOIN city
WHERE contains(city.city_shape, trip.start_location)
AND trip.datestr = '2016-09-07'
```
Spatial Join: The Problem

**Objective:** Associate all trips with city_id for a single day.

```sql
SELECT trip.trip_id, city.city_id
FROM trip JOIN city
WHERE contains(city.city_shape, trip.start_location)
AND trip.datestr = '2016-09-07'
```

Notice that a cross join is involved in the raw query which is prohibitively time consuming.
City Boundary (field simplified_shape)

Geofence shape

OGC format

MULTIPOLYGON (((-122.02681 38.546405, -122.037459
38.580381, -122.041148 38.568211, -122.044402 36.580456,
-122.044148 36.591354, -122.042748 36.586627, -122.034248
36.609673, -122.023636 36.620443, -122.020127 36.622798,
-122.020406 36.624959, -122.019351 36.627761, -122.014173
36.636682, -122.003487 36.646399, -122.001124 36.647704,
-122.068317 36.874953),

...........

...........

-123.073063 37.662593, -123.074921 37.690066, -123.086057
37.70075, -123.09285 37.713688, -123.094801 37.714067,
-123.104921 37.712648, -123.115575 37.714045, -123.126114
37.71673, -123.132597 37.719393, -123.156757 37.73784,
-123.162751 37.743533, (-123.169985 37.75593, -123.172926
37.7838, -123.173763 37.771831)))

...
# Trip City Association Spatial Cross Join

<table>
<thead>
<tr>
<th>trip_uuid</th>
<th>request_lng</th>
<th>request_lat</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-43.9243121</td>
<td>-19.8797076</td>
</tr>
<tr>
<td>2</td>
<td>116.5552567095159</td>
<td>39.89848122355758</td>
</tr>
<tr>
<td>3</td>
<td>-95.438458</td>
<td>29.956148</td>
</tr>
<tr>
<td>4</td>
<td>-77.046441</td>
<td>38.900678</td>
</tr>
<tr>
<td>5</td>
<td>18.4137834</td>
<td>-33.9300258</td>
</tr>
<tr>
<td>6</td>
<td>-0.04861950790666</td>
<td>51.51702180533201</td>
</tr>
<tr>
<td>7</td>
<td>-87.6249684</td>
<td>41.88166650000001</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>n</td>
<td>-70.7489728</td>
<td>-33.4640432</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>city_id</th>
<th>simplified_shape</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>MULTIPOLYGON (((-122.02681 ...</td>
</tr>
<tr>
<td>3</td>
<td>POLYGON ((3.4920327663 ...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>m</td>
<td>POLYGON ((-87.813338 ...</td>
</tr>
</tbody>
</table>

Time complexity (n*m) = 10M trips x 1K cities
= 10B operations ~ 1 week computation time
Spatial Join: Solution

- Use Generated UDFs which uses a geospatial index and avoid cross joins

```sql
SELECT trips.id, getCityId(trips.request_location)
FROM trips
```

We Build either quadtree or r-tree indexes dynamically
Magellan - A self serve Geospatial Service

- Geospatial field for indexing
- A set of UDFs supporting efficient relationship/distance query

Magellan RPC Service

- Collect data and build index for the geospatial field.
- Embed data and index into a pre-defined UDF template.
- Compile and register the output UDFs in Hive.

UBER | Data
Magellan - A self serve Geospatial Service

**Request JSON:**

```json
{
  info: {
    user: string, // user name
    queue: string // hadoop queue to use
  },
  index: {
    namespace: string, // database to register UDF
    prefix: string // prefix of UDF names
  },
  source: {
    table: string, // e.g. dwh.dim_city
    keyField: string, // e.g. city_id
    geoField: string, // e.g.
    simplified_shape: string // e.g.
  },
  register: bool // register persistent UDFs
}
```

**Response JSON:**

```json
{
  state: bool, // is successful
  message: string, // message text to return
  jarUrl: string, // UDF jar file location on HDFS
  host: string,
  udfs: [
    {
      udfName: string,
      description: string,
      className: string
    },
    ...
  ]
}
```
What Next for Spatial Processing?

● Extend ingestion pipelines to support spatial-index fields
● Enhance query-engines (Hive, Presto, Spark) to auto optimize on supported index fields
Key Takeaways

- EL from Source to Data Lake
  - Going back to fetch from online sources over and over again is not a good idea especially at a large scale

- Always manage schemas if you have > 1 consumer
  - When an organization scales, you need automated ways to manage lineage & schema evolution to avoid pain

- Abstract Query Engines Access and Use Standards
  - ANSI SQL - Makes swapping query engines later easier
  - Use a gateway to audit, your SRE/Ops will like you for it

- Leverage Open Source Whenever Possible
  - While filling in the gaps, and contributing back!!
Thank you!