BPMN 6053

MANAGEMENT INFORMATION SYSTEM

GROUP ASSIGNMENT

TECHNOLOGY STACK (UBER)

LECTURER: DR. BHARAT BHUSHAN VERMA

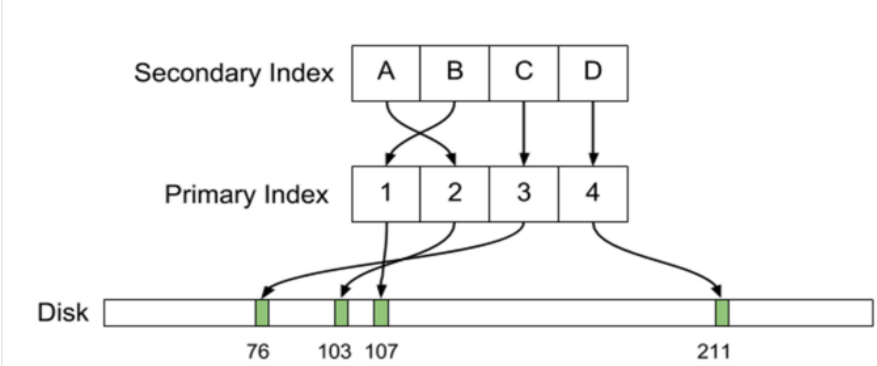
GROUP MEMBERS:

1. SEGAR M MURUGAN (818749)
2. NORAZIMAH BT MOHAMAD TAHIR (819385)
3. ANNAPOORANI A/P GANESON (822413)
4. AYUSHAFIZA ZULKIFLY (816248)
5. NORITA RASLI (822130)

Uber Engineering

1. ***Infrastructure and database***

The early architecture of Uber consisted of a monolithic backend application written in Python that used **Postgres** for data persistence. Since that time, the architecture of Uber has changed significantly to a model of microservices and new data platforms. Specifically, in many of the cases where we previously used Postgres, Uber now use **Schemaless** a novel database sharding layer built on top of MySQL. Here we will explore some of the drawbacks we found with Postgres and explain the decision to build Schemaless and other backend services on top of MySQL.



In addition to explaining some of Postgres's limitations, we also explain why MySQL is an important tool for newer Uber Engineering storage projects such as Schemaless. In this case, Uber found MySQL more favorable for their uses. To understand the differences, we examine MySQL's architecture and how it contrasts with that of Postgres. The postgress has some limitations as following:-

i) Inefficient architecture for writes.
ii) Inefficient data replication.
iii) Issues with table corruption.
iv) Difficulty upgrading to newer releases.

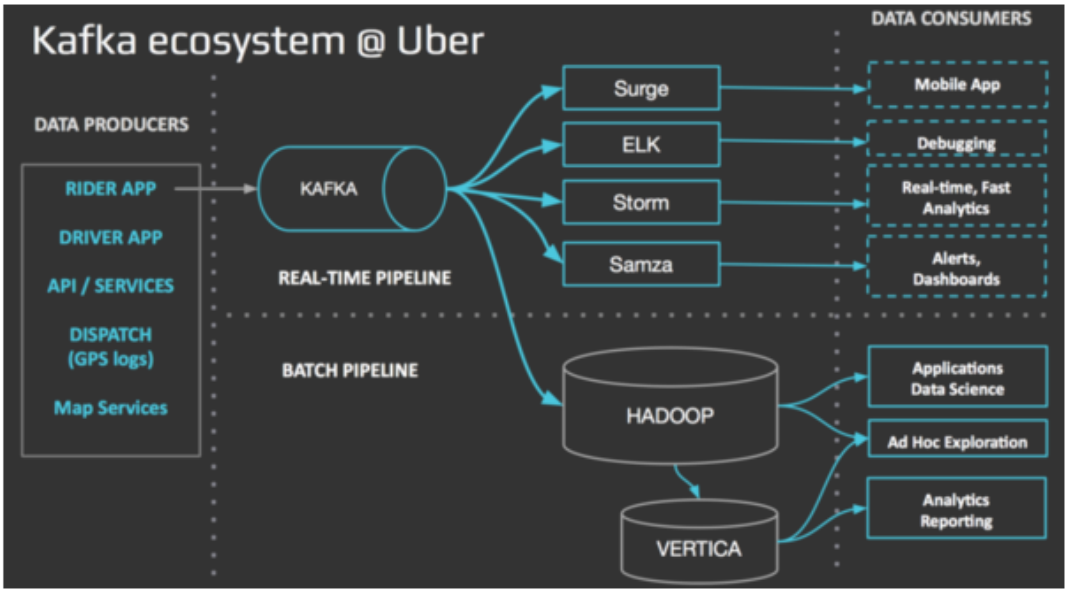
Whereby MySQL supports all the above problems and multiple different replication modes such:

i) Statement-based replication replicates logical SQL statements (e.g., it would literally replicate literal statements such as: UPDATE users SET birth_year=770 WHERE id = 4)
ii) Row-based replication replicates altered row records
iii) Mixed replication mixes these two modes.

2. Logging

Uber services interact with each other and mobile devices, and those interactions are valuable for internal uses like debugging as well as business cases like dynamic pricing. For logging, their use multiple **Kafka** clusters. At Uber, their use Apache

Kafka as a message bus for connecting different parts of the ecosystem. Their collect system and application logs as well as event data from the rider and driver apps. Then make this data available to a variety of downstream consumers via Kafka.



Data in Kafka feeds both real-time pipelines and batch pipelines. The former data is for activities like computing business metrics, debugging, alerting, and dashboarding. The batch pipeline data is more exploratory, such as ETL into Apache Hadoop and HP Vertica.

The **uReplicator**, Uber's open source solution for replicating Apache Kafka data in a robust and reliable manner. This system extends the original design of Kafka's MirrorMaker to focus on extremely high reliability, a zero-data-loss guarantee, and ease of operation. Running in production since November 2015, uReplicator is a key piece of Uber's multi–data center infrastructure. The benefits of uReplicator include:

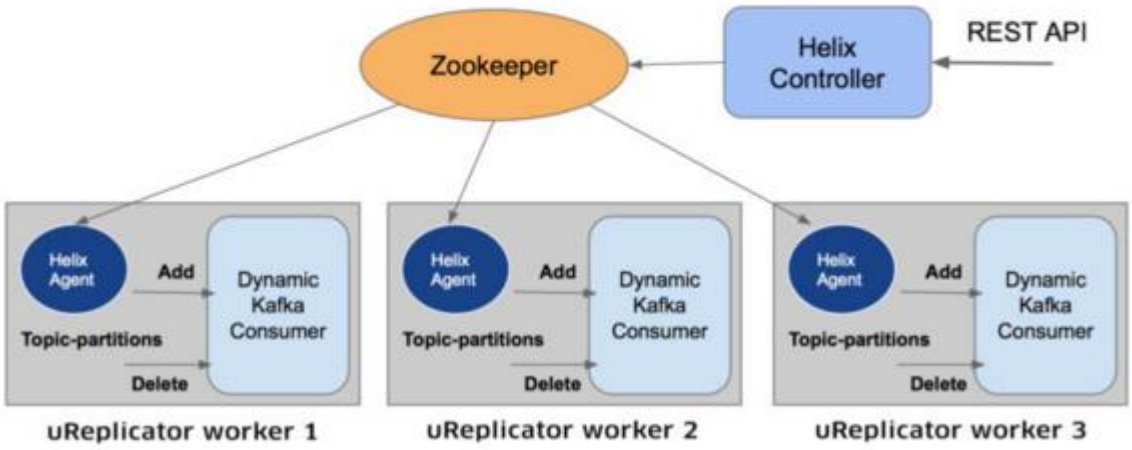
i) Stability: Rebalancing now happens only during startup and when a node is added or deleted. In addition, it only affects a subset of the topic-partitions instead of causing complete inactivity like before.

ii) Easier scalability: Adding a new node to an existing cluster is now much simpler. Since partition assignment is now static, it can intelligently move only a subset of partitions to the new node.

iii) Easier operation: Uber's new mirroring tool supports dynamic whitelisting. Now don't need to restart the cluster when adding/deleting/expanding Kafka topics.

iv) Zero data loss: uReplicator guarantees zero data loss, since it commits checkpoints only after the data has been persisted on the destination cluster.

uReplicator Overview

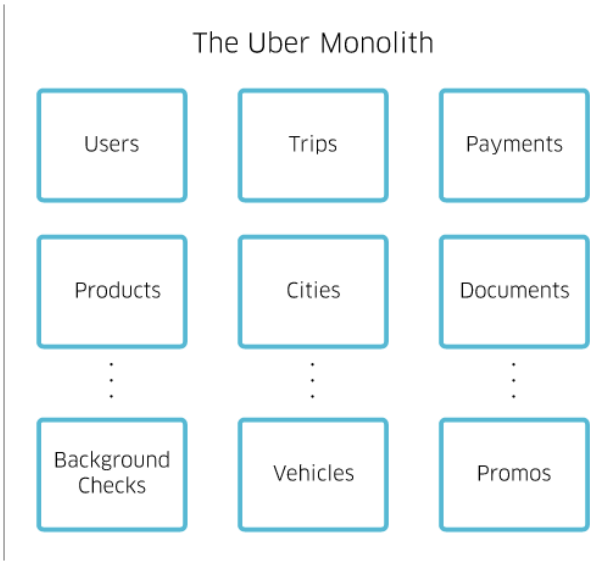


uReplicator's various components work in different ways toward reliability and stability. The Helix uReplicator controller, actually a cluster of nodes, has several responsibilities as:-

i) Distribute and assign topic partitions to each worker process
ii) Handle addition/deletion of topics/partitions
iii) Handle addition/deletion of uReplicator workers
iv) Detect node failures and redistribute those specific topic-partitions

3. Routing and Service Discovery

Uber's **service-oriented architecture (SOA)** makes service discovery and routing crucial to Uber's success. Their decided to follow the lead of other hyper-growth companies such Amazon, Netflix, SoundCloud, Twitter, and others and break up the monolith into multiple codebases to form a **service-oriented architecture** (SOA). Specifically, since the term SOA tends to mean a variety of different things, their adopted a microservice architecture. This design pattern enforces the development of small services dedicated to specific, well-encapsulated domain areas. Each service can be written in its own language or framework, and can have its own database or lack thereof.



Migrating from a monolithic codebase (as above) to a distributed SOA solved many of Uber's problems especially as following:-

i) Obviousness

With 500+ services, finding the appropriate service becomes arduous. Once identified, how to utilize the service is not obvious, since each microservice is structured in its own way. Services providing REST or RPC endpoints (where you can access functionality within that domain) typically offer weak contracts, and in this case these contracts vary greatly between microservices. Adding JSON Schema to a REST API can improve safety and the process of developing against the service, but it is not trivial to write or maintain. Finally, these solutions do not provide any guarantees regarding fault tolerance or latency. There's no standard way to handle client-side timeouts and outages, or ensure an outage of one service does not cause cascading outages. The overall resiliency of the system would be negatively impacted by these weaknesses. As one developer put it, we "converted our monolithic API into a distributed monolithic API". It has become clear that need a standard way of communication that provides type safety, validation, and fault tolerance. Other benefit includes:

-Simple ways to provide client libraries
-Cross language support
-Tunable default timeouts and retry policies
-Efficient testing and development

ii) Safety

The most compelling argument for Thrift is its safety. Thrift guarantees safety by binding services to use strict contracts. The contract describes how to interact with that service including how to call service procedures, what inputs to provide, and what output to expect. In the following Thrift IDL, Uber has defined a service Zoo with a function make sound that takes a string animal name and returns a string or throws an exception.

```
struct Animal {
  1: i32 id
  2: string name
  3: string sound
}

exception NotFoundException {
  1: i32 what
  2: string why
}

service Zoo {
  /**
   *  Returns the sound the given animal makes.
   */
  string makeSound(1: string animalName) throws (
      1: NotFoundException noAnimalFound
  )
```

Adhering to a strict contract means less time is spent figuring out how to communicate with a service and dealing with serialization. In addition, as a microservice evolves we do not have to worry about interfaces changing suddenly, and are able to deploy services independently from consumers. This is very good news for Uber engineers. Their is able to move on to other projects and tools since Thrift solves the problem of safety out of the box.

4. Language

Uber's engineers primarily write in Python, Node.js, Go, and Java. We started with two main languages: Node.js for the Marketplace team, and Python for everyone else. These first languages still power most services running at Uber today.

Node.js

"Node.js is particularly well-suited to writing systems that have all their state in memory," said Kris Kowal, a Software Engineer at Uber. "They do not have to externalize the concerns of a distributed system. As a consequence, the systems can be more available, and they can respond more quickly to requests by eliminating the reading/writing and the serialization of state into a database.

There are three core strengths that make Node.js a particularly good fit for Uber.

- Node.js handles asynchronous I/O requests with a non-blocking, single-threaded event loop. It is particularly well-suited to distributed systems that make a lot of network requests.
- Node.js - and JavaScript in general - is excellent for quick iteration; programs can be inspected and errors can be addressed on the fly without requiring a restart, so developers can publish and deploy new code constantly.
- The active open source community continuously optimizes the technology; it gets better, all the time, practically on its own.

Python

With Python, you could just use the built-in eval() function. However, that introduces a security loophole. Instead, we took a whitelist approach and used the Python AST module to parse an expression into an abstract syntax tree. We defined the handlers of nodes in the tree to protect ourselves further. Dangerous operations that can be used to gain access to the base class (e.g., attribute access) are not implemented and therefore impossible.

With this approach, we not only made Predicate-eval more secure, but also safer to use. Did you know that the expression, None < 1, evaluates to true in Python? Consider this rule: *if feature < 1, then ban the user*. If the feature happens to be missing for one reason or another, then the user would mistakenly be banned. Predicate-eval disallows this kind of unsafe evaluation.

We also introduced some user-defined functions in Predicate-eval, like lower() and upper() for strings. This way, analysts don't need to import those functions.

Since Predicate-eval builds on top of Python, Mastermind also needs to be written in Python. With Python GIL, we don't get access to multiple cores and can't run rules in parallel. So we did some optimization, like caching parsed abstract syntax trees in memory. Now, running 300 complex rules takes only 30 milliseconds.

Building uber language using Python suits our purpose well because:

- The target users of Predicate-eval are analysts (though engineers use it sometimes too), so the syntax needs to be straightforward.
- Python is a dynamic language, which enables parsing and evaluating a string of expressions at runtime. This empowers users to write rules in front-end, and make it available without restarting servers.
- Uber Engineering has many Python-oriented engineers, so there is no learning curve for them to pick up Predicate-eval.

Example Usage of python

Request a Ride

```
# Get products for location
response = client.get_products(37.77, -122.41)
products = response.json.get('products')

product_id = products[0].get('product_id')

# Get upfront fare for product with start/end location
estimate = client.estimate_ride(
    product_id=product_id,
    start_latitude=37.77,
    start_longitude=-122.41,
    end_latitude=37.79,
    end_longitude=-122.41,
    seat_count=2
)
fare = estimate.json.get('fare')

# Request ride with upfront fare for product with start/end location
response = client.request_ride(
    product_id=product_id,
    start_latitude=37.77,
    start_longitude=-122.41,
    end_latitude=37.79,
    end_longitude=-122.41,
    seat_count=2,
    fare_id=fare['fare_id']
)

request = response.json
request_id = request.get('request_id')

# Request ride details from request_id
response = client.get_ride_details(request_id)
ride = response.json

# Cancel a ride
response = client.cancel_ride(request_id)
ride = response.json
```

Although Node.js and Python was the real-time marketplace team's primary programming language , Uber also use Go as it met their needs for the following reasons:

- High-throughput and low-latency requirements. Geofence lookups are required on every request from Uber's mobile apps and must quickly (99th percentile < 100 milliseconds) answer a high rate (hundreds of thousands per second) of queries.

- CPU intensive workload. Geofence lookups require CPU-intensive point-in-polygon algorithms. While Node.js works great for our other services that are I/O intensive, it's not optimal in this use case due to Node's interpreted and dynamic-typed nature.

- Non-disruptive background loading. To ensure we have the freshest geofences data to perform the lookups, this service must keep refreshing the in-memory geofences data from multiple data sources in the background. Because Node.js is single threaded, background refreshing can tie up the CPU for an extended period of time (e.g., for CPU-intensive JSON parsing work), causing a spike in query response times. This isn't a problem for Go, since goroutines can execute on multiple CPU cores and run background jobs in parallel with foreground queries.

While Java takes advantage of the open source ecosystem and integrates with external technologies, like Hadoop and other analytics tools. Go uber efficiency, simplicity, and runtime speed.

5. Telemetry

In simple word, telemetry refers to the process of recording and transmitting the readings of an instrument. Telemetry is an automated communications process by which measurements and other data are collected at remote or inaccessible points and transmitted to receiving equipment for monitoring. The word is derived from Greek roots: *tele* = remote, and *metron* = measure. Systems that need external instructions and data to operate require the counterpart of telemetry, telecommand.

Although the term commonly refers to wireless data transfer mechanisms (e.g., using radio, ultrasonic, or infrared systems), it also encompasses data transferred over other media such as a telephone or computer network, optical link or other wired communications like power line carriers. Many modern telemetry systems take advantage of the low cost and ubiquity of GSM networks by using SMS to receive and transmit telemetry data.

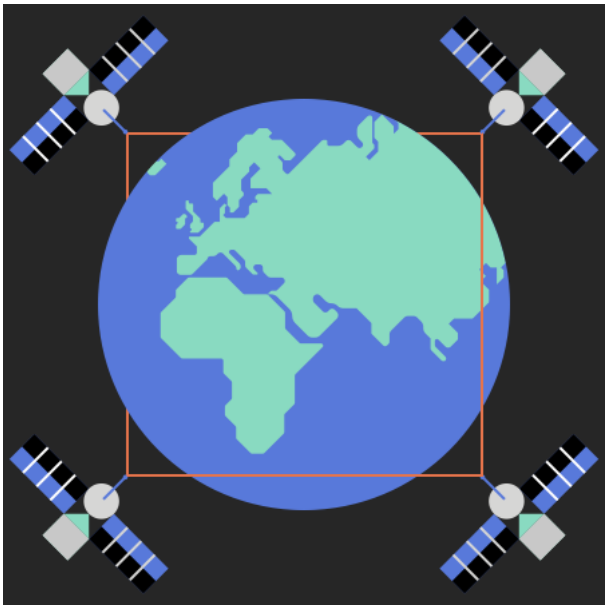
A telemeter is a device used to remotely measure any quantity. It consists of a sensor, a transmission path, and a display, recording, or control device. Telemeters are the physical devices used in telemetry. Electronic devices are widely used in telemetry and can be wireless or hard-wired, analog or digital. Other technologies are also possible, such as mechanical, hydraulic and optical.

Uber developed M3 in Go to collect and store metrics from every part of Uber Engineering (every server, host service, and piece of code). M3 is Uber's proprietary built metrics platform built entirely in NYC and Graphite was the metric system used at Uber. Graphite was used for a long time, however it had poor resiliency, clustering, efficiency, no replication and Extremely high operational cost to expand capacity. It provides a Graphite compatible query interface and it's own query interface M3QL. The limitations of Graphite expressions are Read inside-out - opposite the flow of execution, Grouping by position in path and not enough flexibility. M3QL is a computation language for filtering and grouping by tags, it is pipe based.

After we collect the data, we look for trends. We built dashboards and graphs by modifying Grafana to more expressively contextualize information. Every engineer watching a dashboard tends to care about data in a particular location or region, around a set of experiments, or related to a certain product. We added data slicing and dicing to Grafana. Grafana is a tool for beautiful monitoring and metric analytics & dashboards for Graphite, InfluxDB & Prometheus and More.

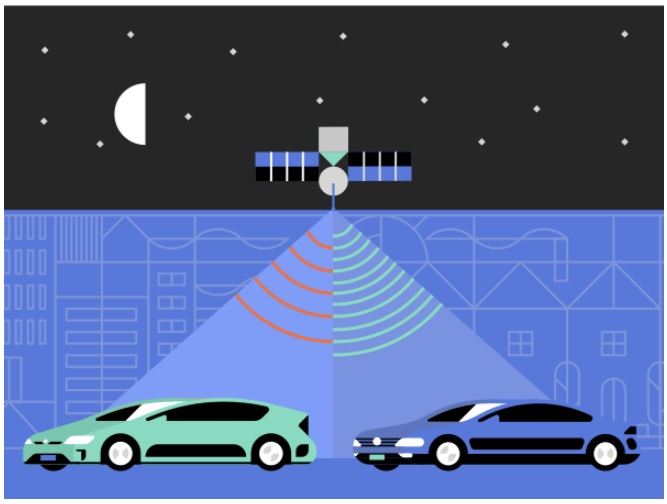
How Uber Engineering Increases Safe Driving with Telematics? Across the globe, nearly 1,250,000 people die in road crashes each year. At Uber, they determined to decrease this number by raising awareness of driving patterns to their partners. In fact, an entire team at Uber focuses on building technology to encourage safer driving**.** On Uber Engineering's Driving Safety team, they write code to measure indicators of unsafe driving and help driver partners stay safe on the road. We measure our success by how much we can decrease car crashes, driving-related complaints, and trips during which we detect unsafe driving. Uber use harsh braking and acceleration as indicators of unsafe driving behavior. Harsh braking is highly correlated to unsafe behaviors like tailgating, aggressive driving, and losing focus on the road. For example, research from Progressive, a car insurance provider, has shown that harsh braking is a leading indicator for predicting future crashes. To detect these indicators of unsafe driving in the first place, we start with a few simple engineering problems.

How Do uber Measure Speed? Before measure abrupt vehicle movement, uber need to measure speed. And to understand speed, they have to understand how GPS works. Put simply, GPS is a system of 24 active satellites that orbit the Earth. The GPS receiver derives its position by determining its distance from at least four satellites.



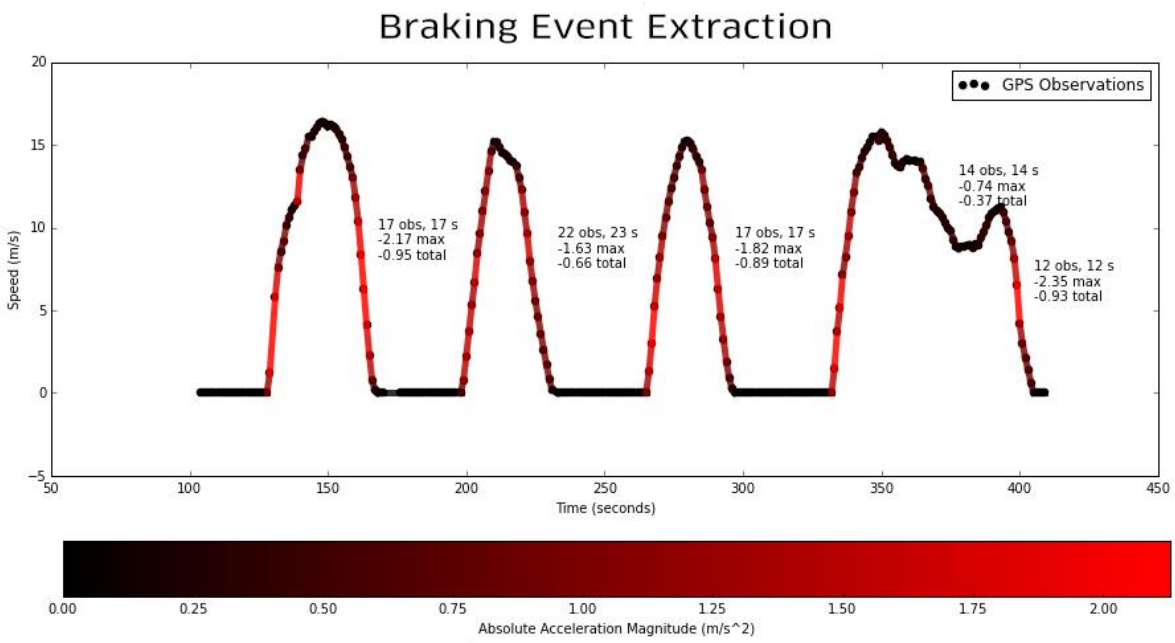
The simplest way of deriving speed from position is by measuring the difference between two consecutive positions. If you know a location is x_1 at time t_1, and x_2 and time t_2, the average speed between those locations is $\frac{x_2 - x_1}{t_2 - t_1}$. This value will approach the true speed as the frequency of measurements increases. While simple to implement, this method depends on GPS positional accuracy, which can be unreliable in urban environments, particularly around tall buildings. Uber get a more accurate measurement of speed by using the Doppler shift, which occurs when a signal's transmitter moves relative to its receiver. Fire truck sirens often illustrate the Doppler shift; the transmitter is the siren, and the receiver is your eardrum. The perceived pitch of the siren increases as the truck moves toward you, and decreases as the vehicle moves away.

GPS receivers on driver partner phones work in a similar way. The receiver (that is, the phone) is either moving toward or away from a satellite. The receiver's velocity can be accurately derived from the difference between the expected signal's frequency and its actual one. GPS can also take a measure of speed by looking at the rate at which the waves that carry the GPS signal change (this is called time-difference carrier positioning).



Measuring Brakes and Accelerations. The next step is to derive brakes and accelerations from vehicle speed. Acceleration is defined as the rate of change of velocity. Therefore, once we measure the vehicle's speed, we can determine the magnitude of the acceleration by calculating the derivative. To start, the Uber Data team extracted a large set of GPS data into a hosted Jupyter notebook with the goal of transforming any arbitrary window of GPS data into a feature vector comprised of summary statistics. We did this by first extracting acceleration and braking events from the time series GPS data, and then by computing summary statistics on the observed events.

The process of extracting braking events from time series speed data is easiest visualized:

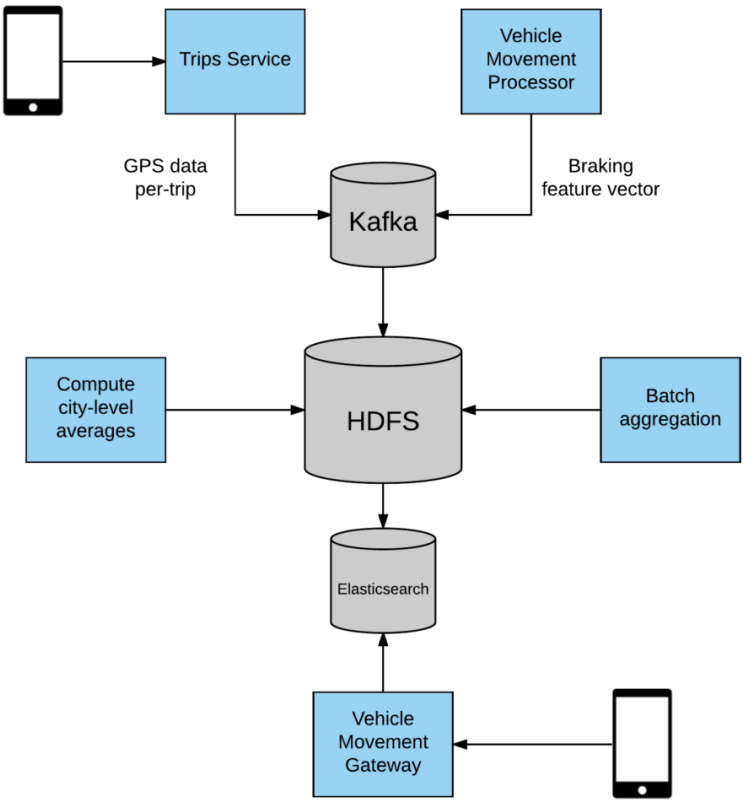


Once they had a series of braking and acceleration events, they computed a variety of descriptive statistics such as:

- The fraction of braking events exceeding 2 m/s^2 .
- The fraction of braking events exceeding just over 3 m/s^2 (7 mph), the threshold set by Progressive for a "hard brake" event.
- The maximum, 90th percentile, and median magnitude of all accelerations.

Feature Engineering. So, how do Uber make sense of these features? In particular, they wanted to know which ones reliably indicated unsafe driver behavior. Fortunately, uber had a large corpus of training data: rider feedback: obtained a set of positive labels from rider feedback, trips with low passenger ratings indicating dangerous driving behavior, and with this set trained a basic machine learning model to validate that their feature set had predictive power on bad rider experiences.

Processing Data at Scale. **Finally, we had to put these learnings in production and efficiently process GPS data at Uber's scale:**



Driver partner phones submit GPS data to the Trips Service, and partner phones receiving telematics data will do so via the Vehicle Movement Gateway. The above diagram shows how partner phone data flows through our architecture. We process and store GPS data from

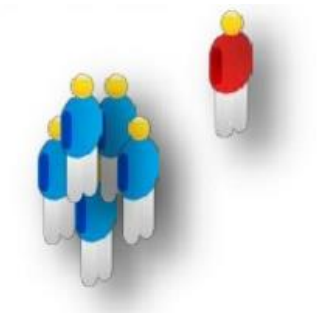
trips in our **Trips Service**. Trip data is then published to a Kafka topic and consumed by many other internal services, one of which is our Vehicle Movement Processor. This service produces a feature vector of driving behavior (num_hard_brakes, peak_accel_magnitude, etc) to yet another Kafka topic to be consumed by more services. All data from Kafka eventually lands into HDFS for long-term storage. We can run batch analysis from our HDFS cluster using tools like Hive and Spark. For example, we can compute daily city-level averages for hard brakes. Then, we index this data with our Elasticsearch cluster for low-latency reads and expose a simple API through the Vehicle Movement Gateway.

This architecture has a number of advantages:

- **The architecture is *fault tolerant*.** Each service is deployed across multiple hosts in multiple data centers, and each data store is distributed in nature.
- **The architecture *scales horizontally*.** We monitor the performance of each component in the architecture and can easily add more nodes if we experience high load.
- **The architecture is *flexible*.** Any service can start consuming from one of our Kafka topics without compromising the health of the system.

6. Anomaly Detection

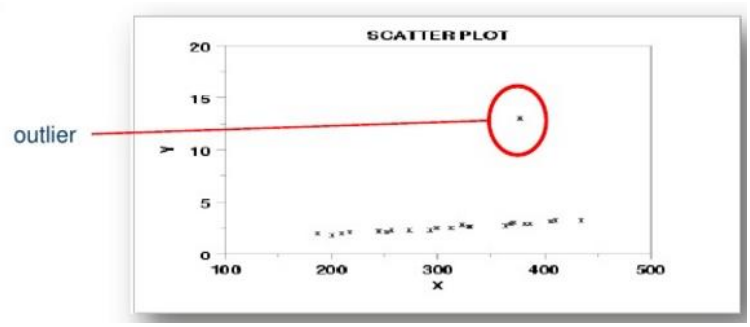
Anomaly detection is the identification of data points, items, observations or events that do not conform to the expected pattern of a given group. These anomalies occur very infrequently but may signify a large and significant threat such as cyber intrusions or fraud. Intrusion detection of attack on computer systems and network. Anomaly detection are process find object that are different from most other objects which also known as outliers. On scatter plot data, they lie far away from other data points.



It’s also known as ;

a) Deviation detection which means anomalous objects have attribute values that deviate significantly from the expected or typical attribute values.

b) Exception mining because anomalies are exceptional in some sense.



Anomaly detection is heavily used in behavioral analysis and other forms of analysis in order to aid in learning about the detection, identification and prediction of the occurrence of these anomalies.

Anomaly detection is also known as outlier detection or data mining tools.

Anomaly detection is mainly a data-mining process and is used to determine the types of anomalies occurring in a given data set and to determine details about their occurrences. It is applicable in domains such as fraud detection, intrusion detection, fault detection, system health monitoring and event detection systems in sensor networks. In the context of fraud and intrusion detection, the anomalies or interesting items are not necessarily the rare items but those unexpected bursts of activities. These types of anomalies do not conform to the definition of anomalies or outliers as rare occurrences, so many anomaly detection methods do not work in these instances unless they have been appropriately aggregated or trained. So, in these cases, a cluster analysis algorithm may be more suitable for detecting the microcluster patterns created by these data points.

What cause anomalies.

1) Data from Different sources

- Someone who committing credit card fraud belong to different class that those people who use credit card legitimately.

Such anomalies are often considerable interest and are focus of anomaly detection, in the field of data mining

An outlier is an observation that differs so much from other observation as to arouse suspicion that is was generated by different mechanism (Hawkins Definition of outliers)

b) Natural Variant

- Many data sets can be modeled by statistical distribution where the probability of a data objects decrease rapidly as the distance of the objects from the center of the distribution increase.

  Most objects are near a center (average object) and the likelihood that an object differs from this average is small

  Anomalies that represent extreme or unlikely varieties are often interesting

c) Data measurement and collection error.

- Error in the data collection or measurement process are another source of anomalies

The goal is to eliminate such anomalies since they provide no interesting information but only reduce the quality of the data and the subsequent data analysis.

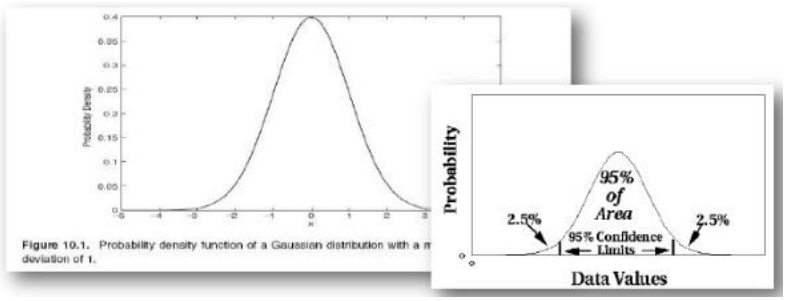
Techniques for anomaly detection include:

One-class support vector machines

- Determination of records that deviate from learned association rules
- Distance-based techniques
- Replicator neural networks
- Cluster analysis-based anomaly detection

Specific techniques for anomaly detection in security applications include:

- Profiling methods
- Statistical methods
    - A model is created for the data and object are evaluated with respect to how well they fit the model
    - Most statistical approach to outlier detection are based on building a probability model distribution model and considering how likely objects are under that model.
    - -Outliers are objects that has a low probability with respect to probability distribution model of the data (Probabilistic Definition of an Outlier)



- Rule-based systems
    - Objects that are in regions of low density are relatively distant from their neighbours and can be considered anomalous.
- Model-based approaches
    - Build a model of the data
    - Anomalies are objects that do not fit the model very well
- Distance based methods
    - Also known as proximity-based techniques
    - Its refer to as distance-based outlier detection technique
    - Anomalous object are those that are distant from most of the other objects

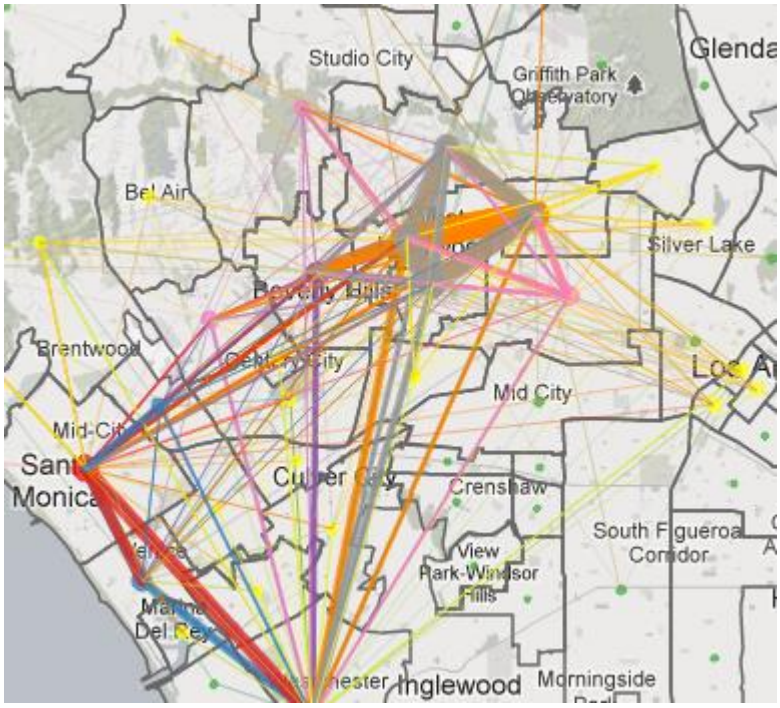
In particular, in the context of abuse and network intrusion detection, the interesting objects are often not *rare* objects, but unexpected *bursts* in activity. This pattern does not adhere to the common statistical definition of an outlier as a rare object, and many outlier detection methods (in particular unsupervised methods) will fail on such data, unless it has been aggregated appropriately. Instead, a cluster analysis algorithm may be able to detect the micro clusters formed by these patterns.

Three broad categories of anomaly detection techniques exist. Unsupervised anomaly detection techniques detect anomalies in an unlabeled test data set under the assumption that the majority of the instances in the data set are normal by looking for instances that seem to fit least to the remainder of the data set. Supervised anomaly detection techniques require a data set that has been labeled as "normal" and "abnormal" and involves training a classifier (the key difference to many other statistical classification problems is the inherent unbalanced nature of outlier detection). Semi-supervised anomaly detection techniques construct a model representing normal behavior from a given *normal* training data set, and then testing the likelihood of a test instance to be generated by the learnt model.

Anomaly strength and weakness;

- Have a firm foundation and build on standard statistical technique
- When there is sufficient knowledge of the data and the type of the test that should be applied, these tests can be very effective.
- There are wide varieties of statistical outliers test for single attributes, fewer options are available for multivariate data.
- Can perform poorly for high-dimensional data.

Uber is a smartphone-app based taxi booking service which connects users who need to get somewhere with drivers willing to give them a ride. The service has been hugely controversial, due to regular taxi drivers claiming that it is destroying their livelihoods, and concerns over the lack of regulation of the company's drivers.



Source for picture: Mapping a city's flow using Uber data

This hasn't stopped it from also being hugely successful – since being launched to purely serve San Francisco in 2009, the service has been expanded to many major cities on every continent except for Antarctica.

Type of anomalies in UBER base on Proximity-base approach.

The basic notation of this approach is straightforward. Its refer to an objects is anomaly if it is distant from most point. More general and more easily applied than statistical approaches. Using the big data model and zoom in anomaly its easier to determine a meaningful proximity measure for sata set that to determine its statistical distribution.

One of the simplest way to measure whether an objects is distant from most point is to use the distance to the *k-nearest neighbor*. The outlier score of an object is given by the distance to its *k-nearest neighbor*. {The lowest value of outlier score is 0 and the highest value is the maximum possible value of the distance function(usually infinity)}

How the system anomalies approach are computing the distance between every pair of data points. The various ways to define outliers too in few method ;

a) Data point for which there are fewer than P neighboring points within a distance D

b) The top n data points whose distance to the other nearest neighbor is greatest

c) The top n data points whose average distance to the other nearest neighbors is greatest.

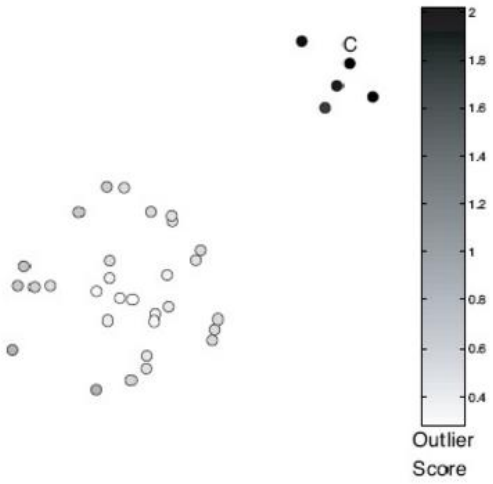


Fig 1: Outlier score based on distance to the fifth nearest neighbor. A small cluster become an outlier.

- The shading of each point indicates its outlier score using values of K=5
- The outlier score can be highly sensitive to the value *k*
- If *k* is too small e.g.,1 then a small number of nearby outliers can cause a low outlier score
- If *k* is too large then its possible for all objects in a cluster that has fewer objects that *k* to become outliers

Using distance or proximity approach in UBER Engineering technology enhance their system ability to the most simplest and easy way approach. The proximity based approach typically take $O(m^3)$time. For large data sets this can be too expensive. The system that UBER had comes with approximately certain cost and how deep data extraction. However using this approach its cover wide data range by region instantly by different densities.

The business is rooted firmly in Big Data and leveraging this data in a more effective way than traditional taxi firms have managed has played a huge part in its success.

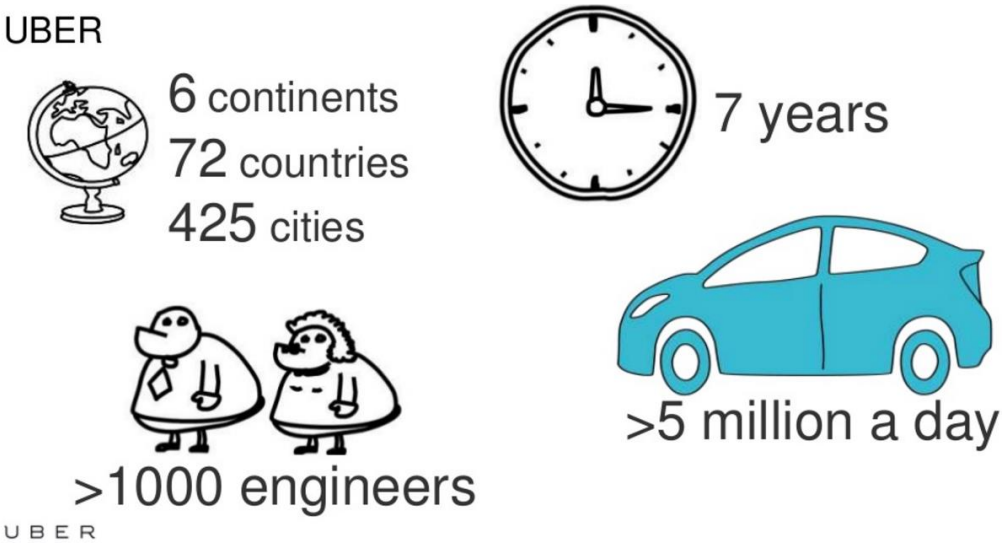
Uber's entire business model is based on the very Big Data principle of crowd sourcing. Anyone with a car who is willing to help someone get to where they want to go can offer to help get them there.

Uber holds a vast database of drivers in all of the cities it covers, so when a passenger asks for a ride, they can instantly match you with the most suitable drivers.

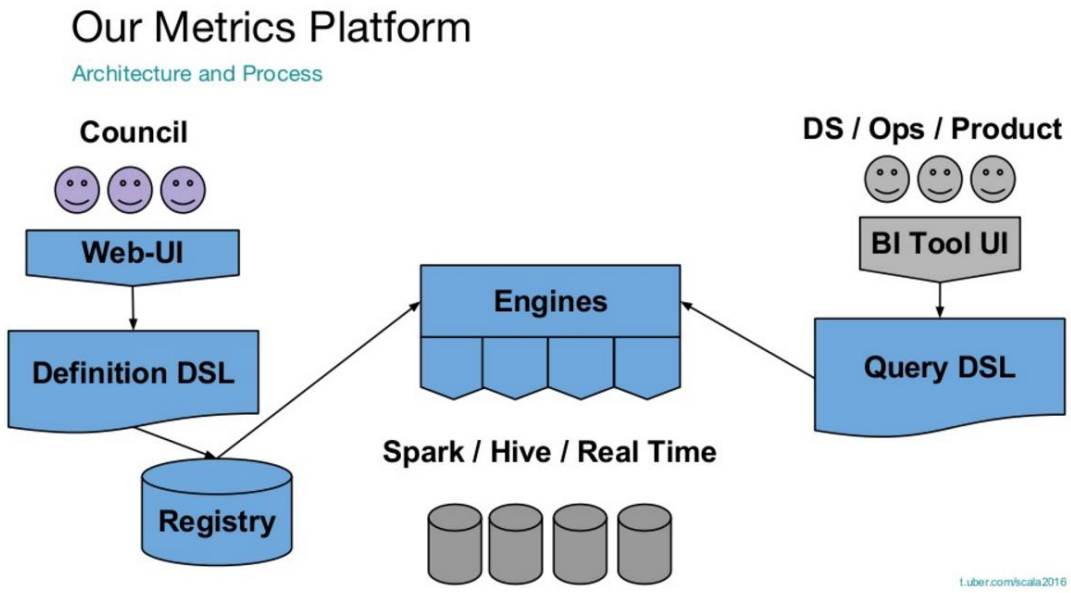
Fares are calculated automatically, using GPS, street data and the company's own algorithms which make adjustments based on the time that the journey is likely to take. This is a crucial difference from regular taxi services because customers are charged for the time the journey takes, not the distance covered.

Acting on Metrics

In UBER business its evolve rapidly base on the need of individual and tech savvy resources that everyone had. The system engineered in for UBER user to have the access and use the facility that provides. UBER system has been use over 425 cities in over 72 countries.



Lots of growth trending double as compare from 2016 and 2015. With the metric platform, UBER manage to deliver the smooth process with technology. Metric designed in aligned with system, reliable and trusted by all user and developer as well as investor in Uber technology.



The metric platforms are easy and powerful, integrated and lightweight process explain in details below:

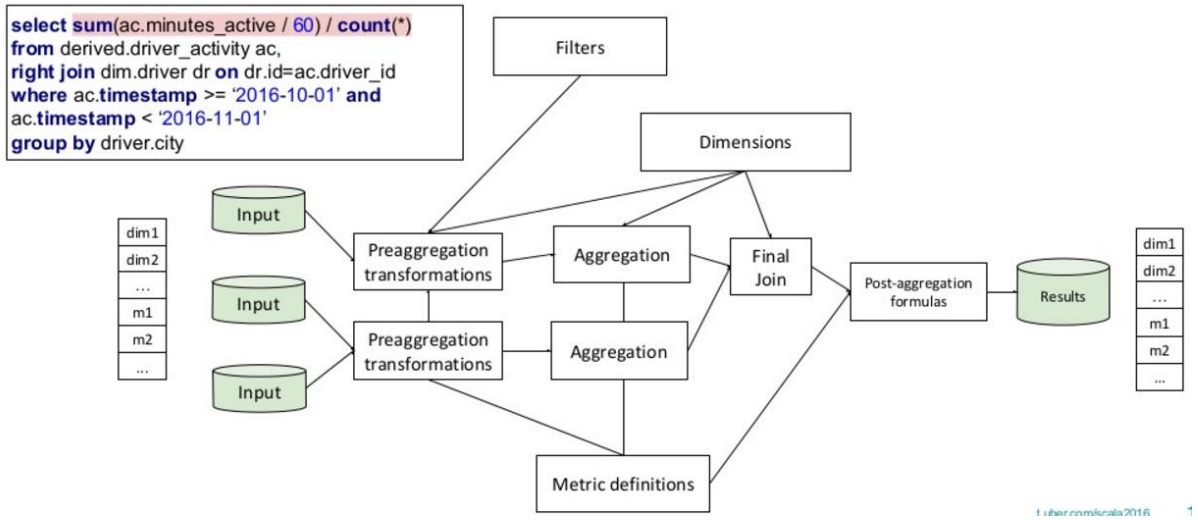
UBER User involve in

- Experimentation
- Product Group
- Financial reporting
- Real time decision making
- Fraud detection

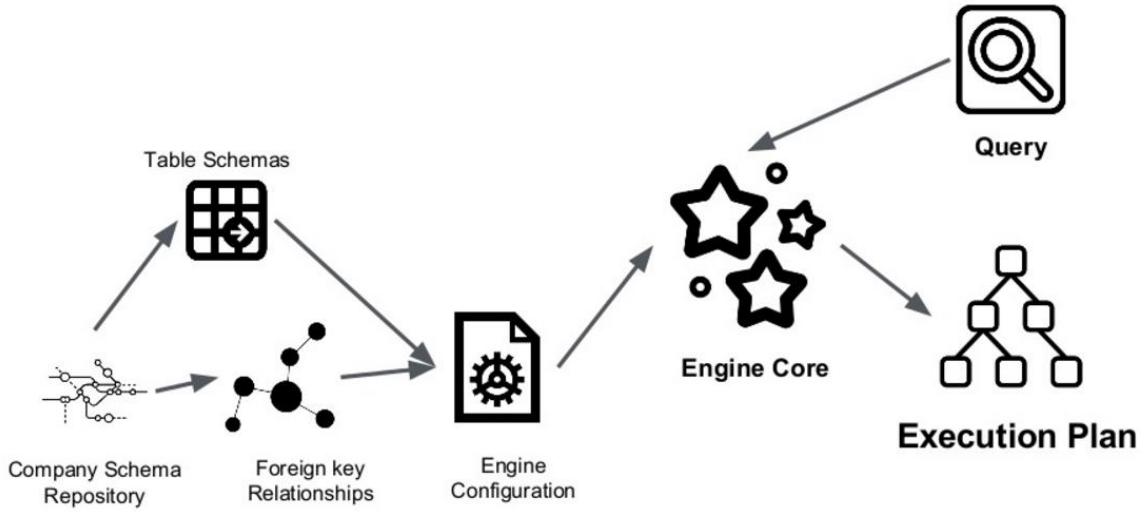
Metric Walkthrough:

Metric	**Hours Active**
English description	Hours spent by drivers logged-in and online in the driver app
SQL	Select sum (ac.minutes_active/60)/count(*) from derived.driver_activity ac,right join dim.driver dr on dr.id=ac.driver_id
Add date	Select sum(ac.minutes_active/60)/count(*) from derived.driver_activity ac,right join dim.driver dr on dr.id=ac.driver_id . **where ac.timestamp>=’2017-10-01 and ac.timestamp<2017-11-01’**
In Which City are you. Eg: Kuala Lumpur	Select sum(ac.minutes_active/60)/count(*) from derived.driver_activity ac,right join dim.driver dr on dr.id=ac.driver_id **join dim.city c on c.id=ac.city_id** where ac.timestamp>=’2017-10-01 and ac.timestamp<2017-11-01’ and c**.name=’Kuala Lumpur’**
Group by experiment treatment	Select sum(ac.minutes_active/60)/count(*) from derived.driver_activity ac,right join dim.driver dr on dr.id=ac.driver_id . join dim.city c on c.id=ac_city_id **Join xp.user_experiment xp on xp.user_id=dr.id** where ac.timestamp>=’2017-10-01 and ac.timestamp<2017-11-01’ and c.name=’Kuala Lumpur’**and xp.experiment_key=’crm_driveronboarding_wcdrip’group by xp.treatment**
Group by driver type	Select sum(ac.minutes_active/60)/count(*) from derived.driver_activity ac, right join dim.driver dr on dr.id=ac.driver_id join dim.city c on c.id=ac.city_id **join model.driver dm on dm.id=dr.id** where ac.timestamp>=’2017-10-01 and ac.timestamp<2017-11-01’ and c.name=’Kuala Lumpur’and xp.experiment_key=’crm_driveronboarding_wcdrip’group by xp.treatment,**model.driver.type**

Anatomy of a metric base on below model:



Also include of Metric = Formula + Query as per above metric table. All the above formula integrated in engine core to develop and complex data source on application usage.



And finally its lightweight process can be managed in Web UI mode.

7. Mapping

Crafting Data-Driven Maps

At Uber, we use maps for everything—visualizing millions of geo data points, monitoring road conditions, and advocating for policy change in cities around the world

Over the last few years we have experienced immense growth. As a result, we have many teams across the organization producing map visualizations for a wide range of needs. We saw a need to create a unified system to guide the creation of consistent, high-quality, data-driven maps.

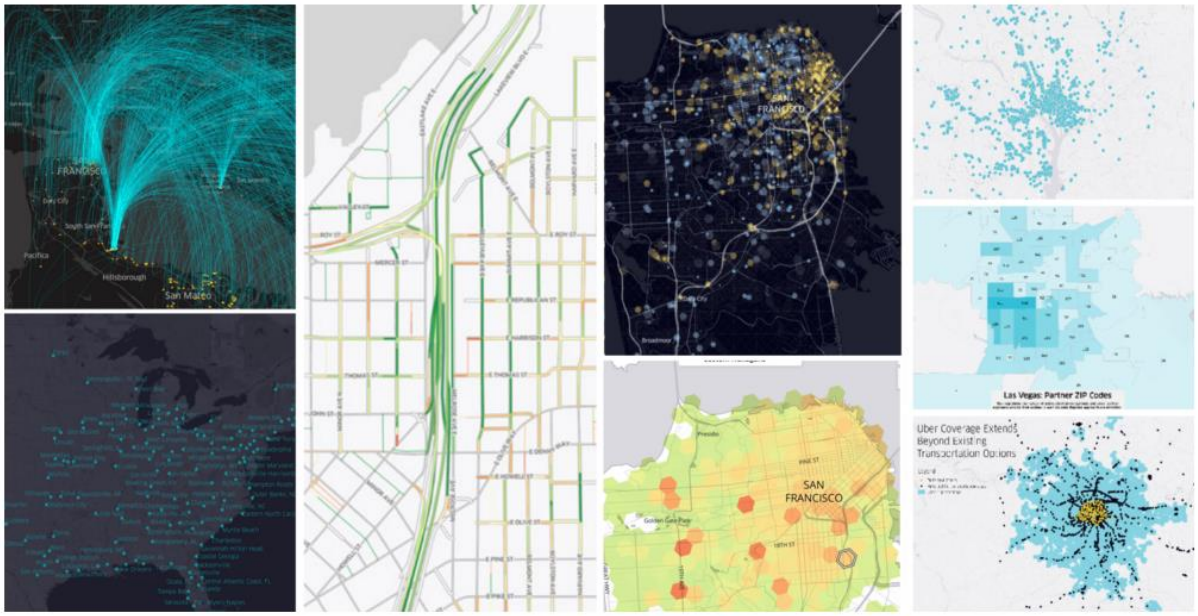
The Challenge(s)

Crafting geospatial visualizations at scale isn't your everyday design task. The complexity and lack of tooling in this space makes it something of a black art. Nevertheless, there aren't many companies in the world that have the rich geo-resources Uber has, so we were excited to explore this creative avenue.

Many teams, many maps

The way we've made maps in the past worked when the company was smaller. Each map was created for specific purpose. However, as the company grew it was no longer enough to have individually great maps. We needed a framework that made sense for a global scale.

We thought it would be useful to have a starting point for teams across the org to scale up and make better maps, faster.



A smattering of maps created in 2016

The image above isn't an exhaustive list, but as you can see, our maps were, *well*—all over the map. Individually, each map is successful, but as a family, they lack consistency. That was a core issue we wanted to address.

Lack of tooling

We immediately cracked open Sketch, eager to get started on this meaty problem. But after staring at the screen blankly for a few minutes, the questions started rolling in.

"*How do we edit vector maps in Sketch?*", "*Should we use Illustrator?*", "*How can we simulate thousands of data points?*", "*Do we need to hand-draw roads?"*
"We can't use any of the tools we know…this is going to take forever…"

It turns out that Sketch, Illustrator and virtually all of the tools we were comfortable with don't have much support for common GIS file formats and common cartographic tasks.

Not being able to use familiar design tools put us a little outside of our comfort zone. We eventually overcame these constraints by **learning newtools and building custom software,** which I'll get to a little later.

Global scale

To further compound the complexity of this problem, we needed our framework to scale up to the needs of **400+ cities around the world**. The vast collection of different geographic features and data types put this design problem into a class all its own.

The Process

Our process was largely characterized by trial and error and by stringing together an unlikely cast of tools to get the job done. Here are some of the highlights of the project.

Creating a foundation

A large portion of our time was spent on defining base map themes. We knew these themes would serve as the foundation of the entire project, so getting them right up front was critical.

Base map styling optimized for data visualization

We **optimized these base maps for data visualization** and focused on three main areas:

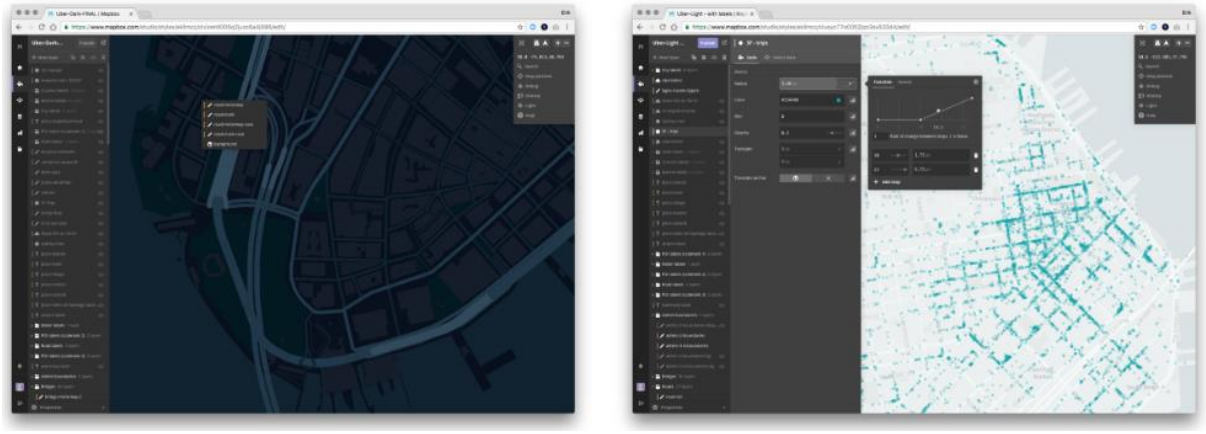
Detail: We fine-tuned the amount of detail you see at any given zoom level. We went with a relatively minimal approach so the data didn't compete with the base map.

Color: Starting with Uber's brand colors, we produced various shades and tones for different map elements. The dark theme was inspired by the night-mode map used in our driver app. The biggest challenge was creating enough contrast between all the different elements that held up for a variety of screens and devices.

Typography: the type system uses our brand typeface (FF Clan) to pull it all together. In the spirit of consistency we went with a relatively simple type ramp to reduce the amount of variation between our maps.

Tools used

Most of the heavy lifting during this phase was done in the the amazing Mapbox Studio. This web-based tool makes it relatively painless to select all the features on a map and set their fill colors, outlines, icons, etc. It also has awesome support for setting styles based on zoom level.



Maxbox Studio – best in class map style editor

Bring on the data

Once we had a first cut of our base maps, we began adding data layers. Luckily for us, sourcing the data wasn't hard; we used our internal systems to generate massive files containing all the geo data we needed. With the data in hand we started with our two most commonly used visualizations:

Scatter plots & Hex Bins

The scatter plots show individual data points, while the hex bins are great for showing density when there are too many overlapping points. The tricky part was finessing the appropriate sizing, opacity, and stroke thickness for points and hexes at different zoom levels.

Scatter plots and Hex bins showing concentration of Uber trip activity

We take our data privacy and security very seriously at Uber. For most public (and many internal) use cases we prefer hex bins or some type of clustering aggregation to obscure any potential privacy concerns that come with showing individual data points.

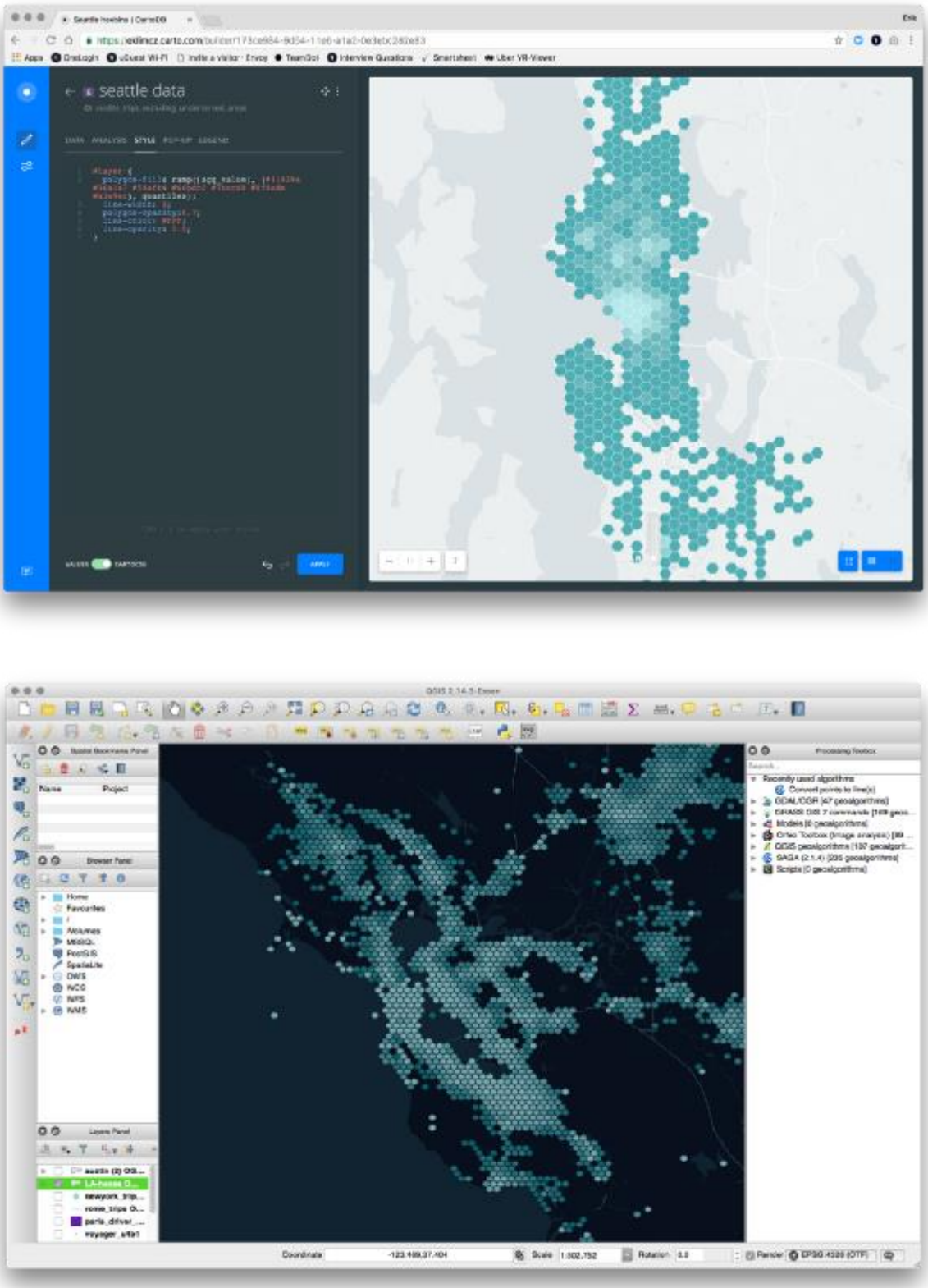
Color scales

Another component of our framework is color. Our internal teams perform extensive data analysis and rely on maps to help them make decisions. Color scales help with finding outliers, trends and ‘needles in the haystack’.

We use one of our core brand colors (Aqua) as the primary hue in most cases, but some datasets required a **sequential color scale** to accent the extents (highs and lows). We also created a **diverging color scale** that is great when you want to accent the mean of your dataset and expose data that significantly ‘diverge’ from the norm.

Tools used

While Mapbox provided everything we needed to style the map itself, we found Carto to have better support for creating custom data-driven styles (although I believe this is coming to Mapbox soon). Carto is also more flexible about the types of data you can upload for visualization. When things got really intense, we had to lean on QGIS for custom filtering, projections and manipulation.



Carto (Left) used for data-driven styling and QGIS (right) used for deeper analysis and manipulation

Taking it a step further

At this point we were happy with overall creative direction, but it was tedious and slow to create a single map visualization. Next we needed a way to accelerate the process and scale it up.

Trip lines

Trip lines are one of the signature visualizations shown in our CEO's TED Talk, and featured in a variety of publications. The challenge is that the actual trip path is not baked into the raw data. It needs to be **generated in code based on an array of latitude and longitude points**. We were hoping there would be a "Generate lines" checkbox in one of our new-found tools, but that wasn't the case.

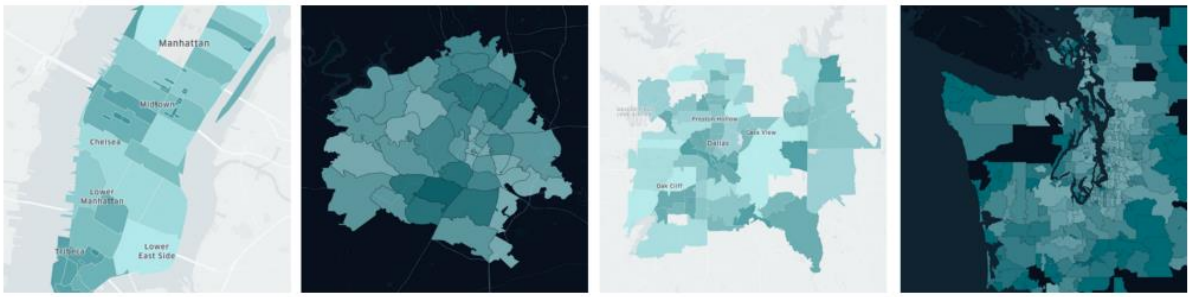
Trip line visualizations showing unique features of different cities

Trip line visualizations showing unique features of different cities

To generate the lines, we created a custom tool with JavaScript, D3.js, and the Mapbox directions API. This tool allowed us to take our massive CSV files and generate geo-based path data with a single drag and drop. Now we can generate 50,000 individual trip paths in about three minutes, and then import them into CARTO or QGIS for styling.

Choropleths

Choropleths help visualize how a metric or value varies across a geographic area. For this case, we used US postal codes as our geographic boundaries and infused various datasets to create the color variation.



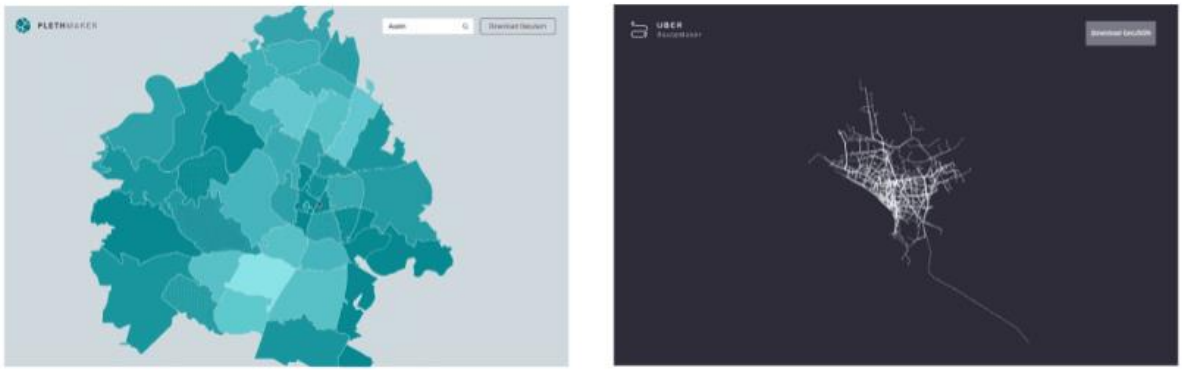
Choropleths based on postal codes generated in D3.js

Choropleths based on postal codes generated in D3.js

Choropleths and Hex Bins are both polygonal in nature, so we gave them the same visual treatment. The biggest struggle with this type of visualization was finding geographic boundary files to experiment with.

Custom Tools

We created **custom tooling to facilitate faster styling iterations** and to quickly scale our design decisions to hundreds of locations around the world.



Custom tools written in javascript and d3.js helped us generate complex scenes quickly to enlarge on location.

We found ourselves spending a lot of time sourcing geo boundary files and tracking down trip lines which made it painfully slow to create single map visualization.

These tools helped speed up our workflow by exporting complex scenes with thousands of individually rendered paths directly to GeoJSON and SVG files. We'd then take those files and import them into other tools like Carto, Mapbox, and QGIS for further styling and manipulation.

The Results

Over the course of this project we produced hundreds of maps to fine-tune the details—colors, typography sizes, opacity, line thicknesses and many more. In the end, these guidelines were delivered as an internal framework that teams across Uber can use as a common starting point for creating high-quality, data-driven maps.

Elements from the final framework

Elements from the final framework

Parting thoughts

Though it can be daunting to tackle problems with this scale and complexity, having such an amazing body of data to pull from is one of the greatest advantages of designing at Uber.

For me personally, this project made me feel like a kid in a data-candy store. In the past, finding an opportunity to explore what's possible with this volume of data would have been nearly impossible. Putting this framework in place allowed us to explore many creative avenues and visualize the data in so many new and interesting ways. There was never a dull moment from start to finish.

8. Payment

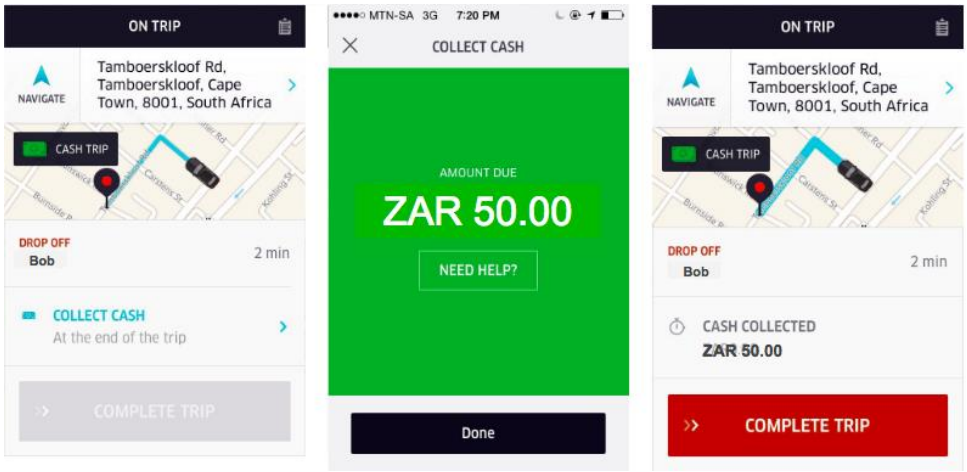
Uber uses a cashless system previously. You can pay via debit or credit card, or use a promo code. This removes any human-to-human cash transfers from the equation, including tips. (Uber takes 25 percent of a driver's fare, which makes for a rather profitable business model.)

When accepting card payments, there are certain requirements that companies must comply with. In the US, these are known as PCI requirements. The Payment Card Industry Data Security Standards (PCI DSS) is a set of requirements designed to ensure that all companies that process, store, or transmit credit card information maintain a secure environment. In effect, this applies to any merchant that has a Merchant ID (MID).

Uber chose to partner with Braintree, one of the leaders in the mobile payment market, to accept card payments. Just for the record, we should mention another great payment system, Stripe, which provides its payment services to Uber's competitor Lyft and to other popular on-demand economy startups such as Postmates and Instacart.

Uber also uses PayPal's Card.io service for credit card scanning on iOS. Card.io allows you to input credit card information by simply holding up your credit card in front of your phone's camera. The app 'sees' the card, reads the numbers, and enters the information for you. With Android, though, you have to manually enter credit card data to link a card to your Uber account.

In addition, PayPal's system has Uber integration, meaning that customers can pay for the taxi service directly from their PayPal accounts (though this is not available globally in all of Uber's markets). You can also split your fare with your friends directly in the app.



Started from 26 May 2016, riders able to pay for trips using Credit Cards and Cssh. As part of the fare reduction experiment they have seen a huge number of people signing up but unable to take a ride because their cards do not work.