SPARK+AI SUMMIT 2018

UBER

Large-scale Feature Aggregation Using Apache Spark

Pulkit Bhanot, Amit Nene
Risk Platform

#Dev1SAIS

Agenda

UBER

- Motivation
- Challenges
- Architecture Deep Dive
- Role of Spark
- Takeaways

Team Mission

UBER

Build a scalable, self-service Feature Engineering Platform for predictive decisioning (based on ML and Business Rules)

- Feature Engineering: use of domain knowledge to create Features
- Self-Service for Data Scientists and Analysts without reliance on Engineers
- Generic Platform: consolidating work towards wider ML efforts at Uber

Sample use case: Fraud detection

UBER

Detect and prevent bad actors in real-time

Payments

Promotions

number of trips over X hours/weeks

trips cancelled over Y months

count of referrals over lifetime

...

Time Series Aggregations

UBER

Needs

- Lifetime of entity
- Sliding long-window: days/weeks/months
- Sliding short-window: mins/hours
- Real-time

Existing solutions

- None satisfies all of above
- Complex onboarding

Warehouse	Streaming aggregations	Indexed databases

None fits the bill!

Technical Challenges

UBER

- Scale: 1000s of aggregations for 100s million of business entities
- Long-window aggregation queries slow even with indexes (seconds). Millis at high QPS needed.
- Onboarding complexity: many moving parts
- Recovery from accrual of errors

Our approach

UBER

One-stop shop for aggregation

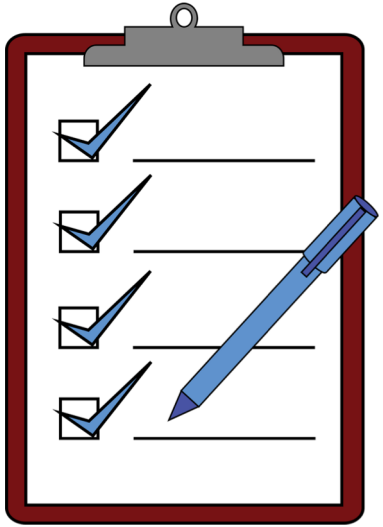
- Single system to interact with
- Single spec: automate configurations of underlying system

Scalable

- Leverage the scale of Batch system for long window
- Combine with real-time aggregation for freshness
- Rollups: aggregate over time intervals
- Fast query over rolled up aggregates
- Caveat: summable functions

Self-healing

- Batch system auto-corrects any errors

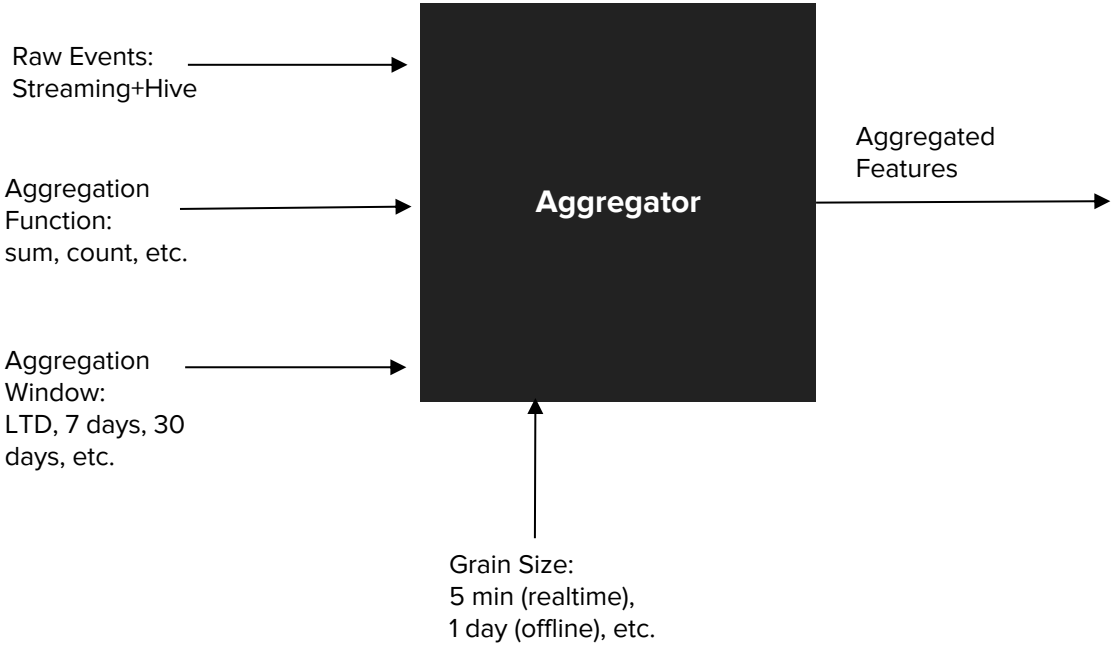


Aggregator as a black box

UBER

Input parameters to black box

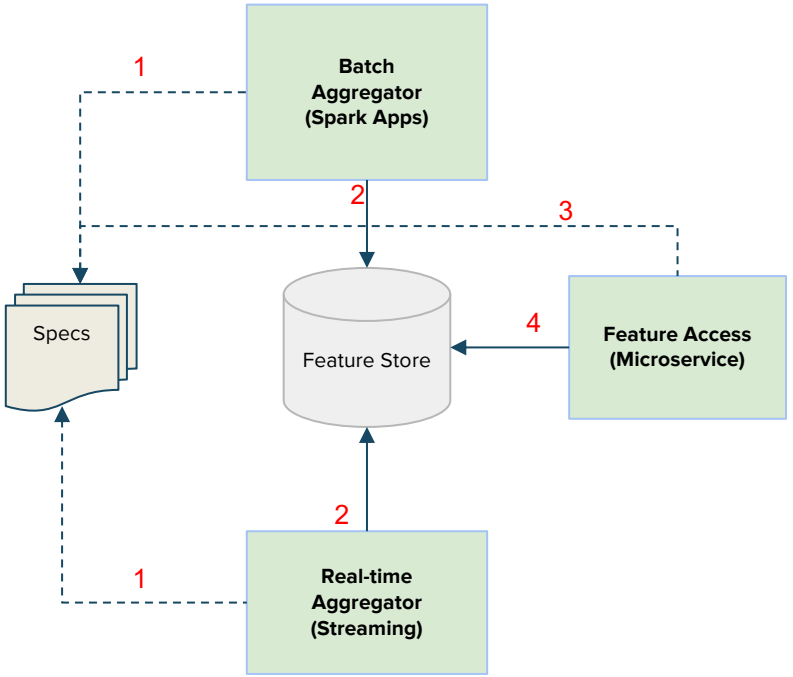
- ➔ Source events
- ➔ Grain size
- ➔ Aggregation functions
- ➔ Aggregation windows



Overall architecture

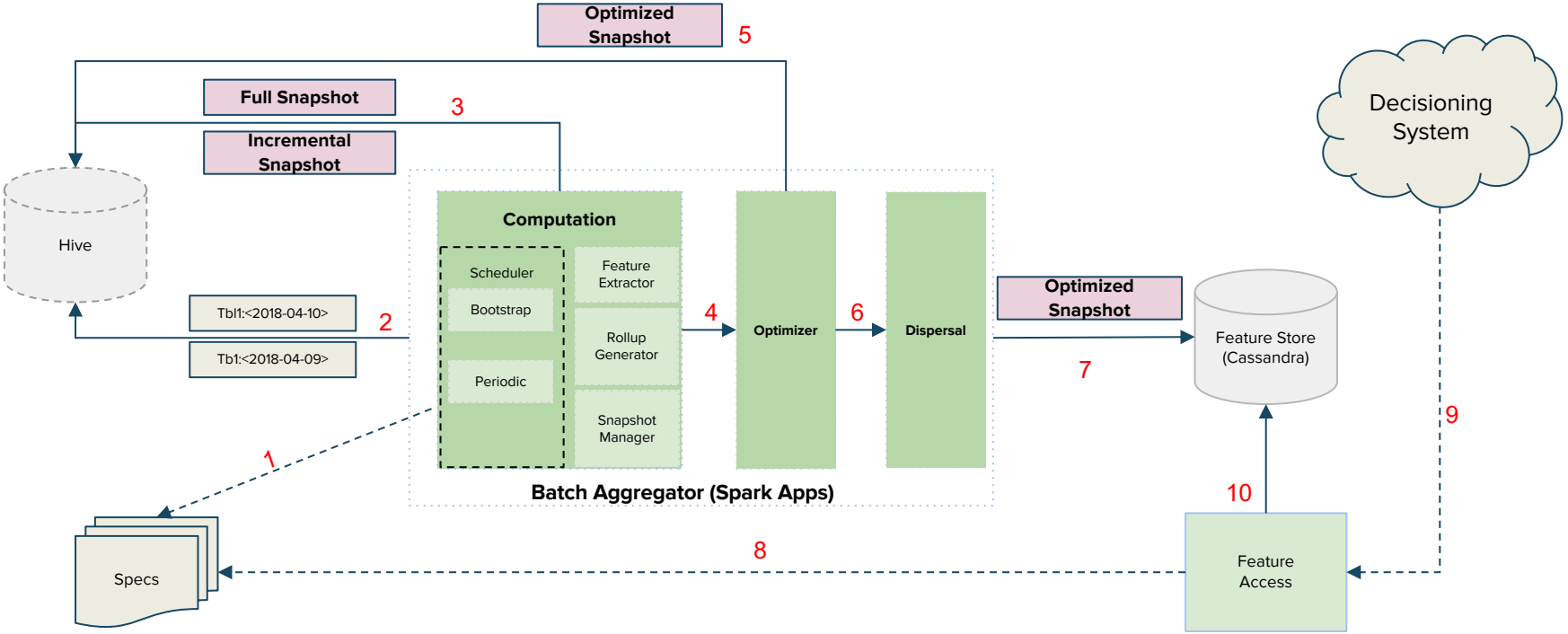
UBER

- Batch (Spark)
  - Long-window: weeks, months
  - Bootstrap, incremental modes
- Streaming (e.g. Kafka events)
  - Short-window (<24 hrs)
  - Near-real time
- Real-time Access
  - Merge offline and streaming
- Feature Store
  - Save rolled-up aggregates in Hive and Cassandra



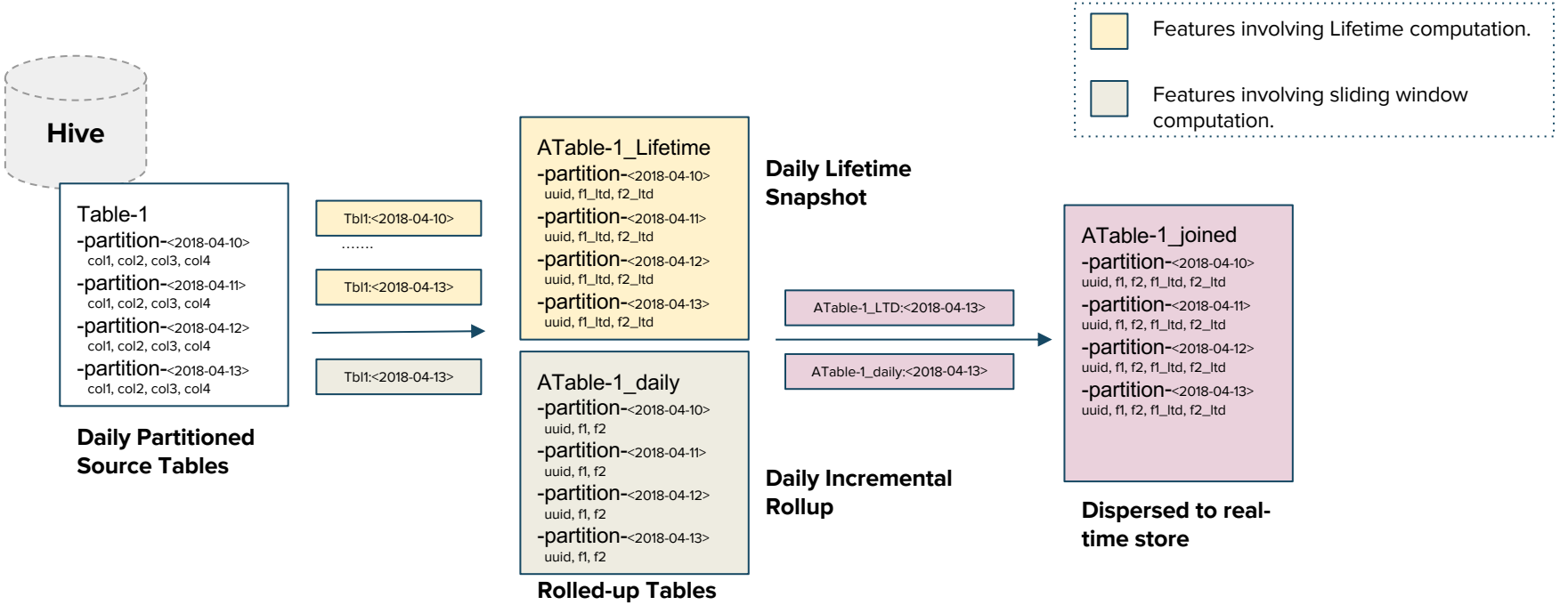
Batch Spark Engine

UBER



Batch Storage

UBER



Role of Spark

UBER

- Orchestrator of ETL pipelines
  - Scheduling of subtasks
  - Record incremental progress
- Optimally resize HDFS files: scale with size of data set.
- Rich set of APIs to enable complex optimizations

e.g of an optimization in bootstrap dispersal

```
dailyDataset.join(
  ltdData,
  JavaConverters.asScalaIteratorConverter(
        Arrays.asList(pipelineConfig.getEntityKey()).iterator())
    .asScala()
    .toSeq(),
  "outer");
```

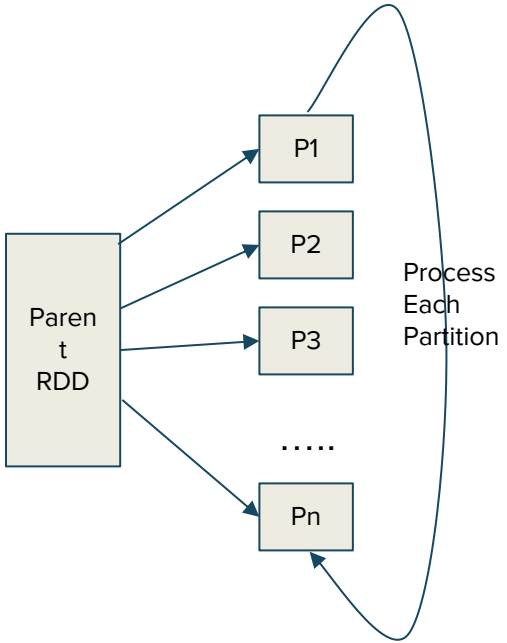
uuid	_ltd	daily_buckets
44b7dc88	1534	[{"2017-10-24":"4"},{"2017-08-22":"3"},{"2017-09-21":"4"},{"2017-08-08":"3"},{"2017-10-03":"3"},{"2017-10-19":"5"},{"2017-09-06":"1"},{"2017-08-17":"5"},{"2017-09-09":"12"},{"2017-10-05":"5"},{"2017-09-25":"4"},{"2017-09-17":"13"}]

Role of Spark (continued)

UBER

- Ability to disperse billions of records
  - HashPartitioner to the rescue

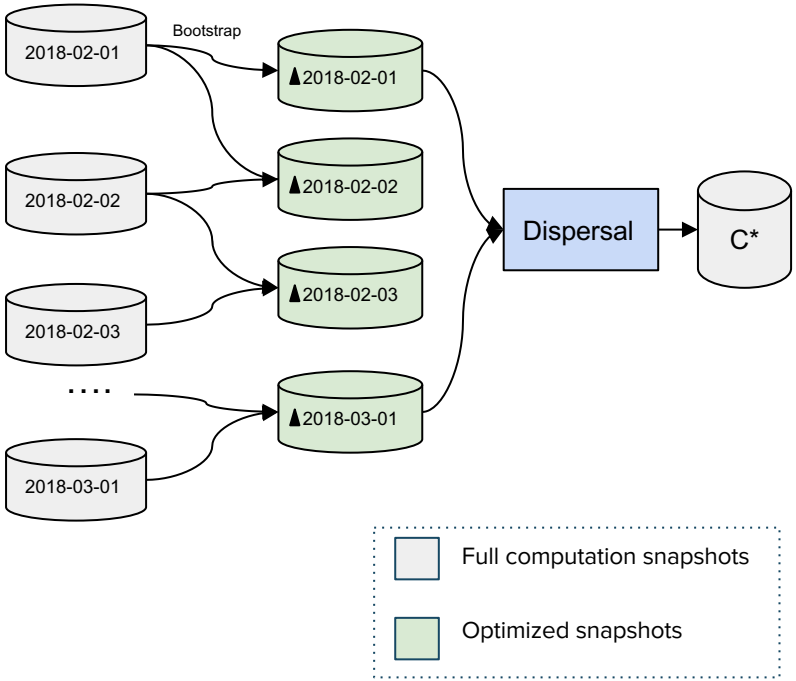
```
//Partition the data by hash
HashPartitioner hashPartitioner = new HashPartitioner(partitionNumber);
JavaPairRDD<String, Row> hashedRDD = keyedRDD.partitionBy(hashPartitioner);

//Fetch each hash partition and process
foreach partition{
 JavaRDD<Tuple2<String, Row>> filteredHashRDD = filterRows(hashedRDD, index, paritionId);
                    raise error if partition mismatch
 Dataset<Row> filteredDataSet =
    etlContext.getSparkSession().createDataset(filteredHashRDD.map(tuple -> tuple._2()).rdd(),
data.org$apache$spark$sql$Dataset$$encoder);
 //repartition filteredDataSet, update checkpoint and records processed after successful
completion.
```



Role of Spark in Dispersal

UBER

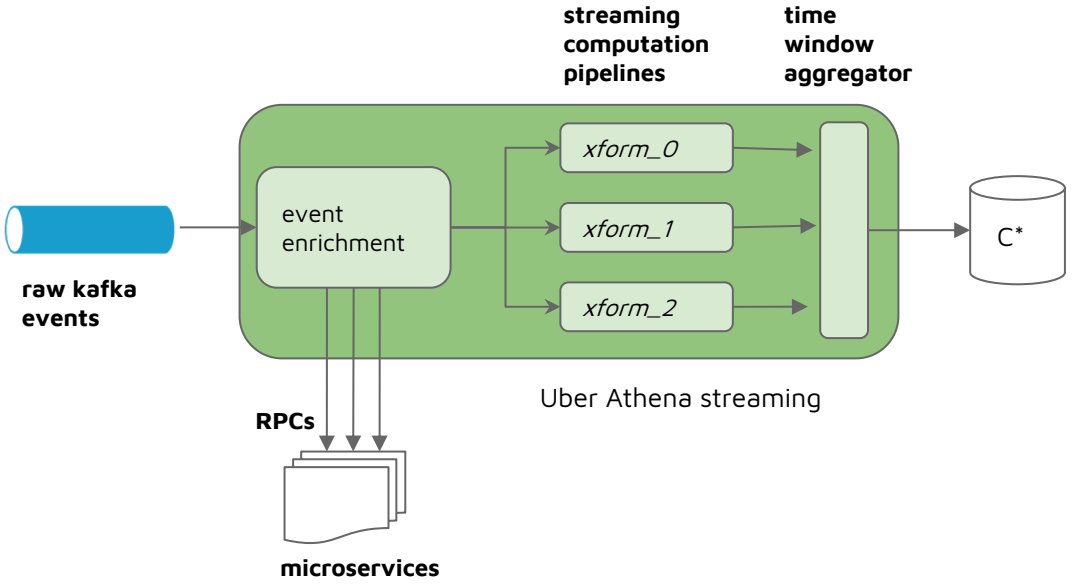
- Global Throttling
  - Feature Store can be the bottleneck
  - *coalesce()* to limit the executors
- Inspect data
  - Disperse only if any column has changed
- Monitoring and alert
  - create custom metrics



Real-time streaming engine

UBER

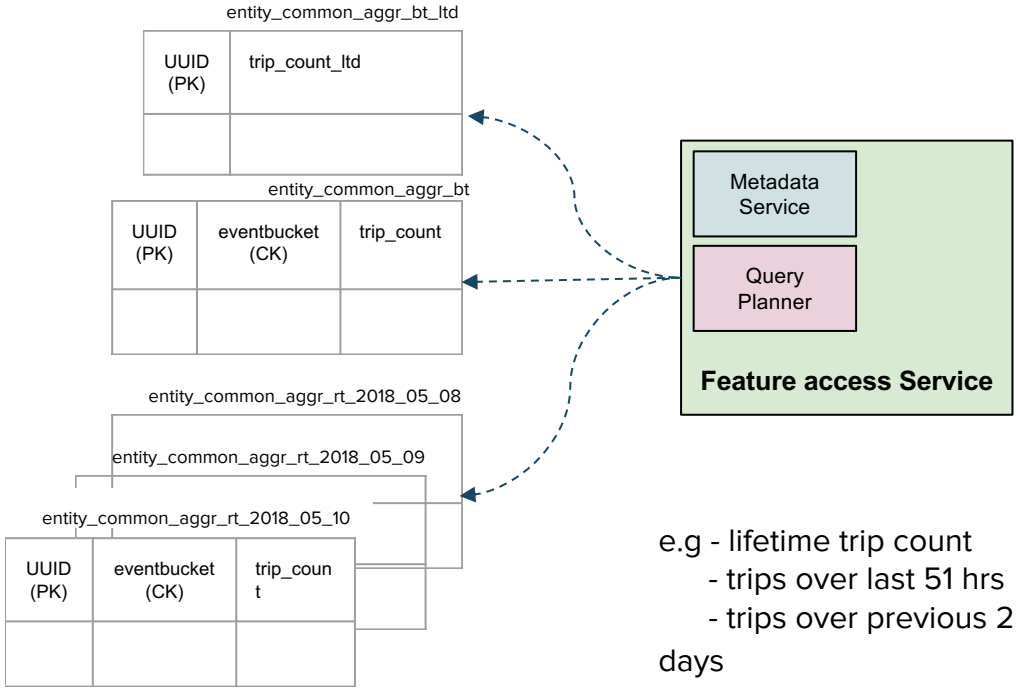
- Real-time, summable aggregations for < 24 hours
- Semantically equivalent to offline computation
- Aggregation rollups (5 mins) maintained in feature store (Cassandra)



Final aggregation in real time

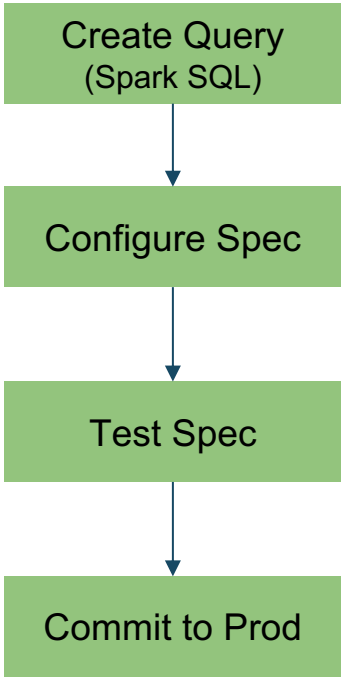
UBER

- **Uses time series and clustering key support in Cassandra**
  - 1 table for Lifetime & LTD values.
  - Multiple tables for realtime values with grain size 5M
- **Consult metadata and assemble into single result at feature access time**



e.g - lifetime trip count
- trips over last 51 hrs
- trips over previous 2 days

Self-service onboarding




```
name: rider.feature_common
owner: abc@uber.com,xyz@uber.com
source: custom
keyColumn:
    - name : uuid
      featureType : string
features:
  buckets: [30M, 2H, 1D, 14D, 30D, LTD]
  definitions:
    - name: trip_count
      operation: count
      featureType: bigint
    - name: trip_amt
      operation: count
      featureType: double
destinationUNS: cassandra.cluster2
destinationType: cassandra
custom:
  realtime:
   kafkaTopic: hp-topic1
   transformClass: com.uber.usecurity.uflow.athena.plugins.dataXForms.Topic1Transform
  offline:
     partitionedHiveTable: dwh.fact_trip
     partitioningColumn: datestr
     dateFormatType: DateTypeWithYYYYMMDD
     dailyQuery: "SELECT uuid, count(uuid) AS trip_count, sum(amount) as trip_amt, datestr
                  FROM database.trip_table
                  WHERE #DATEEXPR#
                  AND status ='COMPLETED'
                  GROUP BY uuid, datestr"
     ltdQuery:   "SELECT uuid, count(uuid) AS trip_count_ltd, sum(amount) as trip_amt_ltd
                  FROM database.trip_table
                  WHERE #DATEEXPR#
                  AND status ='COMPLETED'
                  GROUP BY uuid"
     ltdDuration: 730
```

Machine learning support

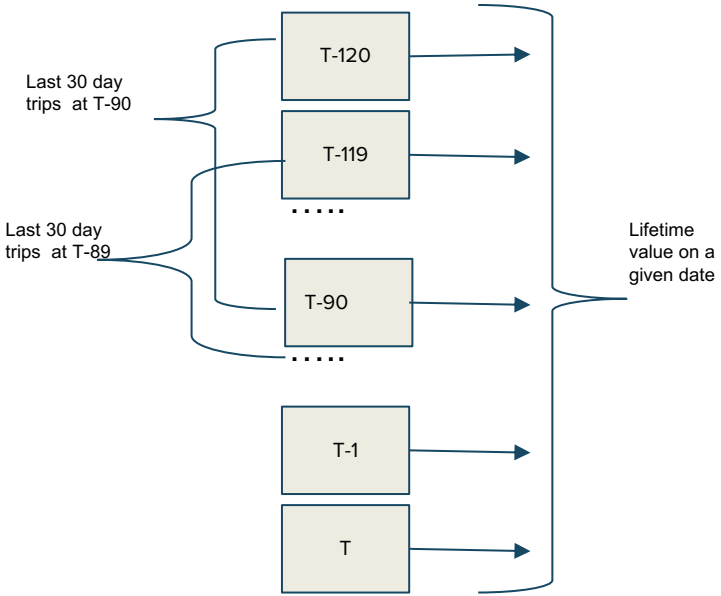
UBER

Backfill Support: what is the value of a feature *f1* for an entity *E1* from T_{hist} to T_{now}

- Bootstrap to historic point in time: T_{hist}
- Incrementally compute from T_{hist} to T_{now}

How ?

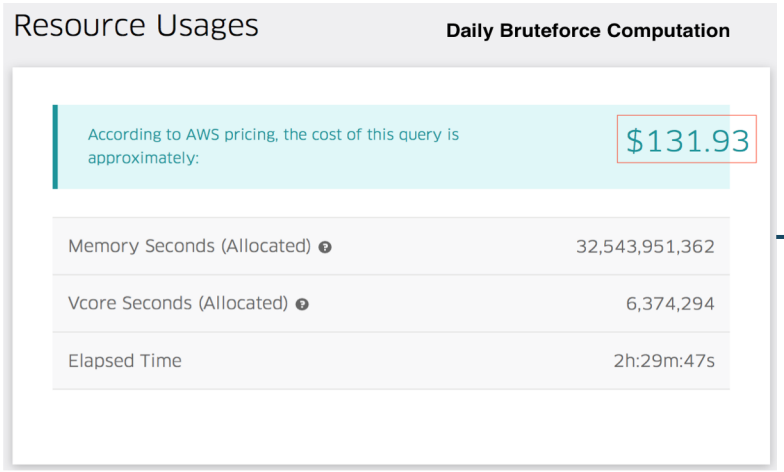
- Lifetime: feature *f1* on T_{hist} access partition T_{hist}
- Windowed: feature *f2* on T_{hist} with window N days
- Merge partitions T_{hist-N} to T_{hist}

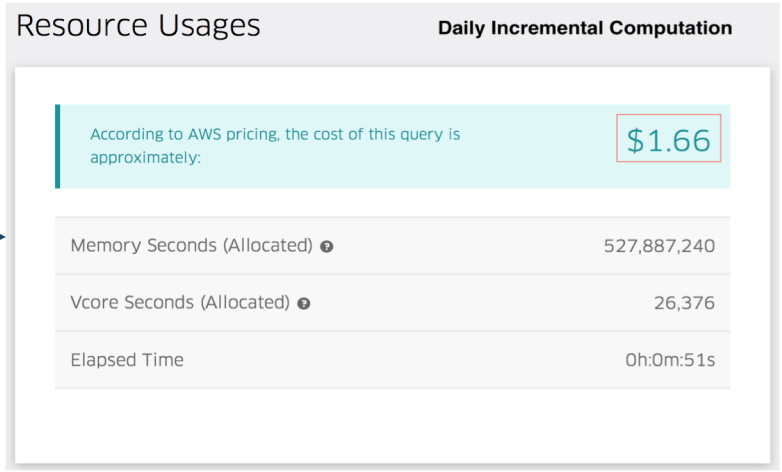


Takeaways

UBER

- Use of Spark to achieve massive scale
- Combine with Streaming aggregation for freshness
- Low latency access in production (P99 <= 20ms) at high QPS
- Simplify onboarding via single spec, onboarding time in hours
- Huge computational cost improvements





UBER

bhanotp@uber.com
anene@uber.com