Neutrino

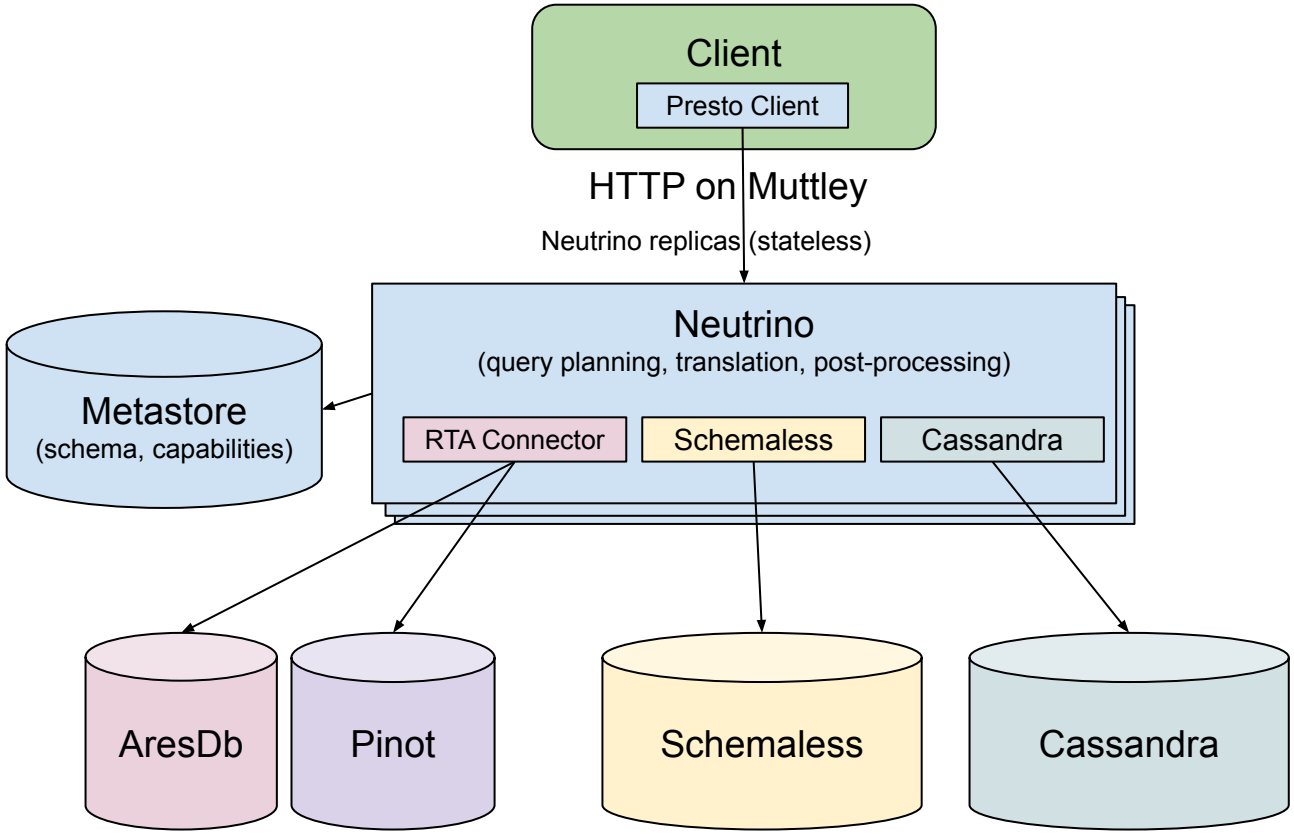
(Super) low latency Presto SQL

Devesh Agrawal, Databricks (formerly Uber)
Bhavani Sudha, Uber

What is Neutrino?

- A federation layer on top of other “online” databases ranging from Apache-Pinot, AresDB, ElasticSearch, Cassandra, Sharded MySQL etc
- Speaks regular Presto SQL
- A regular microservice: POST SQL Query, Get synchronous answer
- Just another deployment of Presto core: same code, different configs

What is Neutrino?



Who uses Neutrino at Uber?

- Custom dashboards used by Uber City Operations:
  - Terra: Map of restaurant hotspots in a city
  - Wisdom: Analysis of incoming mobile bug reports
  - uEconomics: Price surge mismatch at city block level granularity
- What do they need ?
  - Fresh real time data
  - Low latencies. Interactive exploration.
  - Usually Geographic visualization

Why Federation/SQL ?

- Futureproofing: Indirection !
  - Easy to evolve backend storage
  - Switch backend engines based based on query shape
- Onboarding:
  - Everybody (should) know SQL already
  - Its googleable.
- Tooling/Dev experience:
  - Easy to iterate on the SQL query vs code
  - Visualization and other “SQL tools” integration
  - Express joins, transformations in SQL instead of code
  - See data bugs quicker
  - Let the optimizer figure out the right join strategy

Neutrino’s key innovations in Presto

- Aggressive and Complete Pushdown
  - Maximally leverage the underlying engine
  - Push down aggregations, group by, even Joins
  - Leverage and enhance Facebook’s connector plan optimizer framework
- Low latency and Microservice
  - Reduce “Presto Overhead” 800 milliseconds to less than 70 milliseconds
  - Neutrino acts as a synchronous REST microservice
  - Neutrino is stateless and trivially scaleable

Innovation 1: Aggressive Pushdown

- Why ?
  - Pulling raw data from underlying engine is too slow (milliseconds instead of a minute)
  - Does not leverage the underlying engine hardware like GPU for AresDB
- Pushdown implemented as optimizer rules:
  - Runs after other optimizers
  - Uses the new “Connector Optimizer” framework that lets connectors optimize the plan
- Result:
  - Can push aggregations, group by, equi joins and spatial joins.
  - Even complex expressions like approx_distinct, approx_percentile
  - Correctly synthesize the underlying engine query using the pushed plan subtree

Innovation 2: Reducing overhead

- Each Neutrino instance is a coordinator+worker combo
- Overhead stemmed from three main sources:
  - HTTP based Task Assignment b/w coordinator to worker (upto 100 ms)
  - HTTP based Result fetching b/w coordinator to worker (upto 100 ms)
  - Superfluous number of stages for a simple “Scan only” query (upto 50 ms)
  - Unnecessary round trips in assigning splits to tasks (upto 50 ms)
  - Remove resource groups to avoid lock contention (upto 25 ms)

Neutrino deployment results at Uber

- Neutrino overheads range from 10ms to 100ms
  - Remaining time is for underlying engine query
  - Query optimization can take upto 70ms
- Same codebase as regular Presto (Hive) with different configs
- Deployed as regular Java microservice on Uber’s uDeploy
- Stateless and Trivially scaleable: HTTP proxy for load balancing
- Integrates with Uber’s monitoring and logging frameworks

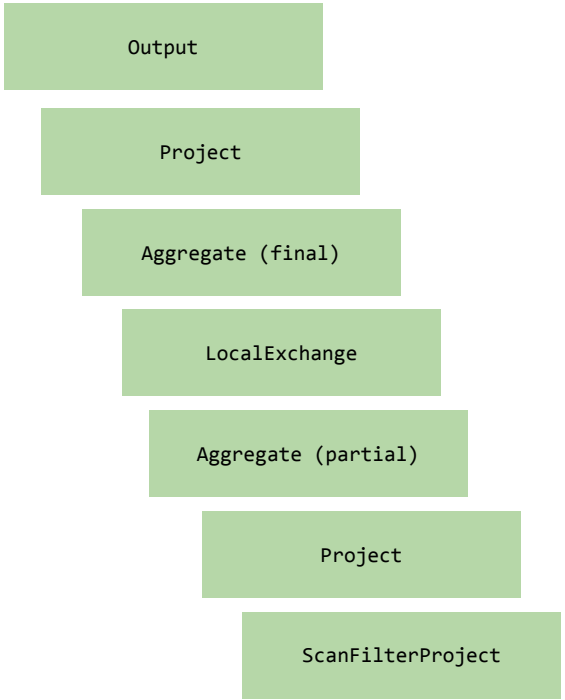
Production experience and lessons

- Used by three ops oriented dashboards:
  - Terra: Live Uber Eats restaurant monitoring and map display
  - Wisdom: Live Uber mobile bug reports and exploration
  - uEconomics: Monitoring surge pricing
- Development in progress
  - uGraph: Uber graph engine leveraging the cassandra connector
  - Neutrino for “SQL on microservices”: Querying services using regular SQL
- Lessons
  - Develop with the customer: Close coordination helped prioritize pushdown features
  - Measuring overhead is hard

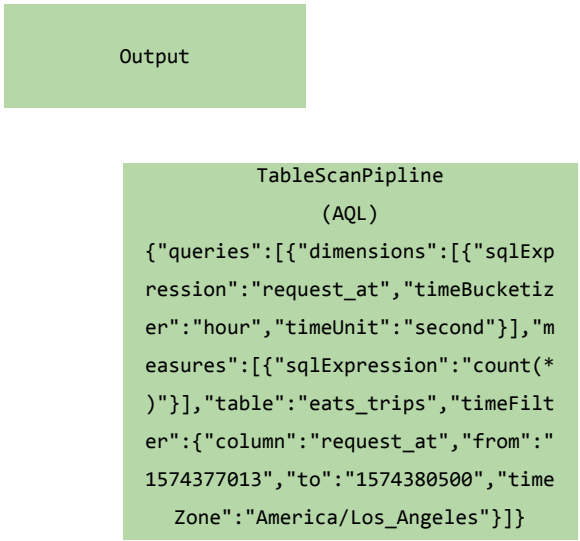
Query

```
explain
select
  date_trunc(
    'hour',
    from_unixtime(request_at, 'America/Los_Angeles')
  ),
  count(*) as completed_trips
from
  rta_staging.rta.eats_trips
where
  request_at between cast(
    to_unixtime(
      at_timezone(now() - interval '1' hour, 'America/Los_Angeles')
    ) as bigint
  )
  and cast(
    to_unixtime(
      date_trunc('hour', at_timezone(now(), 'America/Los_Angeles'))
      + ((minute(now())) / 5) * interval '5' minute
    ) as bigint
  )
group by
  1;
```

Before



After



Open Sourcing plan: Push everything upstream !

- All development on prestodb trunk. No forking.
- Connectors:
  - Pinot Connector: **Already in PrestoDb trunk**
  - AresDB connector: In Progress
- Low latency:
  - Query timeline instrumentation: [PR-13649]
  - Single Stage Plan:
  - Avoid HTTP within local host: In Progress
  - Single shot split assignment:
  - Measuring presto overhead:
- API Changes:
  - Synchronous POST response: [PR-13696]

Outstanding issues and Future directions

- High planning (query optimization) time: Upto 70 milliseconds !
  - Need true query parameterization
  - https://github.com/prestosql/presto/issues/1141
- More connectors:
  - Thrift on Steroids: Seamless and easy SQL over microservices
  - Cassandra, Schemaless, HBase, Elastic Search
- Reducing Latency p99:
  - Making coordinator more push/event based

Thanks to the Presto Community at Facebook !

- Shout out to James, Yi and Saksham
- *Your planner refactoring is really what made all this possible*
- Great to work with!
  - Very supportive and collaborative
  - Quick review of PR
  - Quick engagement and discussion over IM and in person

Questions ?