Presto at LinkedIn and Latency Sensitive Queries

Shardool

Engineer, LinkedIn

Agenda

1	Applications and Usage
2	Presto Query Analysis
3	Query Identification
4	Previous Work
5	Future Plans
6	Q&A

Applications and Usage

Diverse range of applications

- **Reporting** - Understanding business performance and activity
- **Economic Research** - Economics research powered by the data we have
- **Customer Service** - Investigating incorrect behavior on the site
- **Data Products** - Building products for our user powered by data
- **System Engineering** - Analyzing internal system performance

Applications - Clients and Access Patterns

- **Dashboards**
  - Tableau
  - Internal Tools
    - Raptor (Reporting tool)
- **Interactive Query Tools**
- **Applications**
  - Customer service tools
  - Debugging tools

Query Analytics

How to analyze usage across entire range of applications?

- **User growth** – Who's using Presto? Who's new?
- **Tools used** – What clients and tools do users use ?
- **Expensive queries** – Where are the resources going?
- **Resource intensive tables** – Which tables are worth optimizing?
- **Solution** - Event Listener

Query Analytics - Limitation

No streamlined way to address frequent questions referring to specific queries

- Why does my dashboard take longer to load than it used to?
- Why are some critical queries running slower today?
- Can the run-time for a specific query be made more predictable?
- Expecting a sudden jump in application usage, can we plan for it ?

Query Analytics - Takeaways

- Users care deeply about execution characteristics of some of their queries
- These queries are executed repeatedly. Examples -
  - Periodic reports
  - Dashboard loads

Realization

- No generic way to identify different executions of what an end-user considers the ‘same’ query

Query Identification

Opens multiple possibilities for recurring queries

- **Query Specific Analytics** - frequency, Nth percentile execution time
- **Resource Usage** - capacity usage of specific queries
- **Alternate Scheduling** - allowing for ‘better’ execution characteristics for specific queries
- **SLA** - mutually agreed latencies for queries

Query Identification - Application Example

- **Raptor** - Internal Dashboard Tool
  - Large number of users
  - Certain dashboards are very popular
- **Example** - query from a dashboard

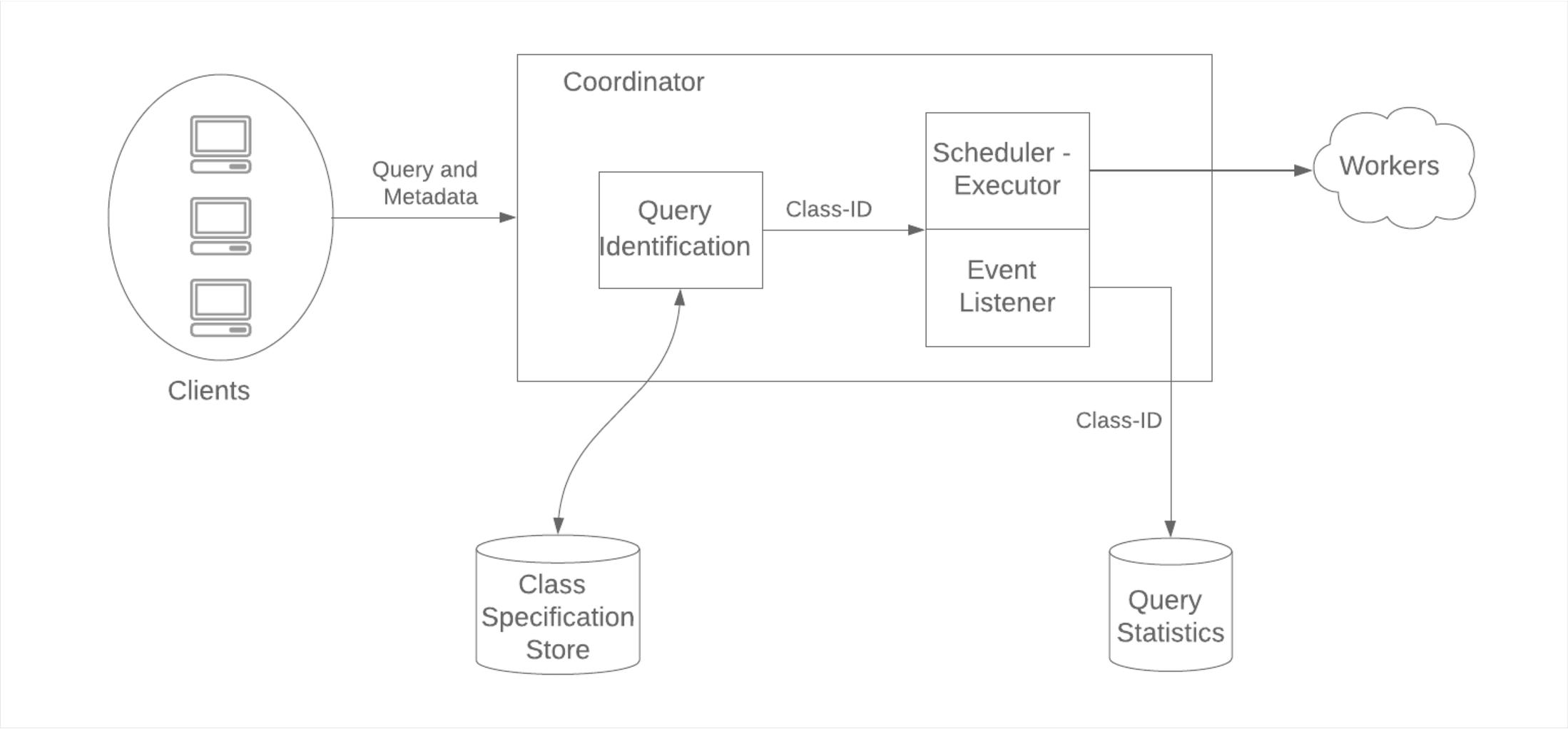
```
select eventdate, count(distinct user)
   from service_events where
   eventdate >= '2019-02-07' and
   eventdate <= '2019-03-08'
   group by eventdate;
```

- **Query Template** - usually such queries are generated from a template by an application
- End-users typically refer to every query from a single template as the ‘same’ query
- Represents a large variety of applications

Query Identification - Considerations

- **Online vs offline** - identifying a query before it starts executing makes it possible to schedule or plan it differently
- **Flexibility of identification** - should allow for some flexibility based on the use-case. For example - if a query is run on 6 days of data instead of a week, then it should still be possible to identify them as ‘same’.
- **Variety of Clients** - work across a diverse set of applications.
  - Internal clients
  - 3rd party BI tools
- **Ease of configuration** - minimize amount of configuration needed for identifying individual queries

Query Identification - Approach



Query Identification - Query Class & Trait

A ‘Query Class’ represents a set of queries which are similar but allows for minor variations which can be configured in advance.

A ‘Query Trait’ defines attributes of a query. A combination of traits can associate the query to a unique class.

Example queries

Query 1

```
select avg(extendedprice) from lineitem
  where shipmode = 'AIR';
```

Query 2

```
select avg(extendedprice) from lineitem
  where shipmode = 'SHIP';
```

Examples of possible traits

- Username
- Application specific identifiers
  - Such as dashboard Ids
- Regex matching query string
- Client Tags

Query Identification - Example 1

Sample Queries

Query 1

```
select avg(extendedprice) from
lineitem where shipmode = 'AIR';
```

Query 2

```
select avg(extendedprice) from
lineitem where shipmode = 'SHIP';
```

Example Traits

Trait 1 - Query Regex Trait

```
select avg(extendedprice) from
lineitem where shipmode = [:alpha:];
```

Trait 2 - Session User

```
price_analysis
```

Trait 3 - Session Property

```
dashboard_id = ‘42’
```

Query Identification - Example 2

Sample Queries

Query 1

```
select shipmode, avg(extendedprice) from
lineitem where shipdate >= date '1998-11-24'
and shipdate < date '1998-12-01' group by
shipmode;
```

Query 2

```
select shipmode, avg(extendedprice) from
lineitem where shipdate >= date '1998-11-17'
and shipdate < date '1998-11-24' group by
shipmode;
```

Example Traits

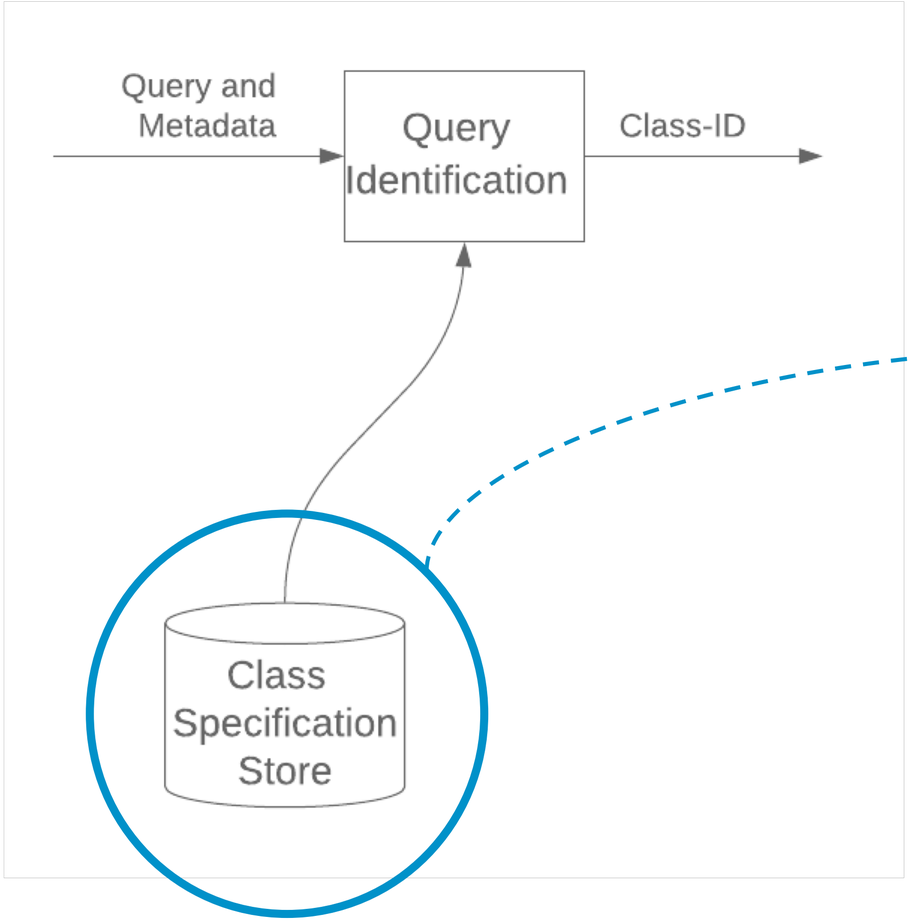
Trait 1 - Query Regex Trait

```
select shipmode, avg(extendedprice) from
lineitem where shipdate >= date [:date:] and
shipdate < date [:date:] group by shipmode;
```

Trait 2 - Session User

```
price_analysis
```

Query Identification



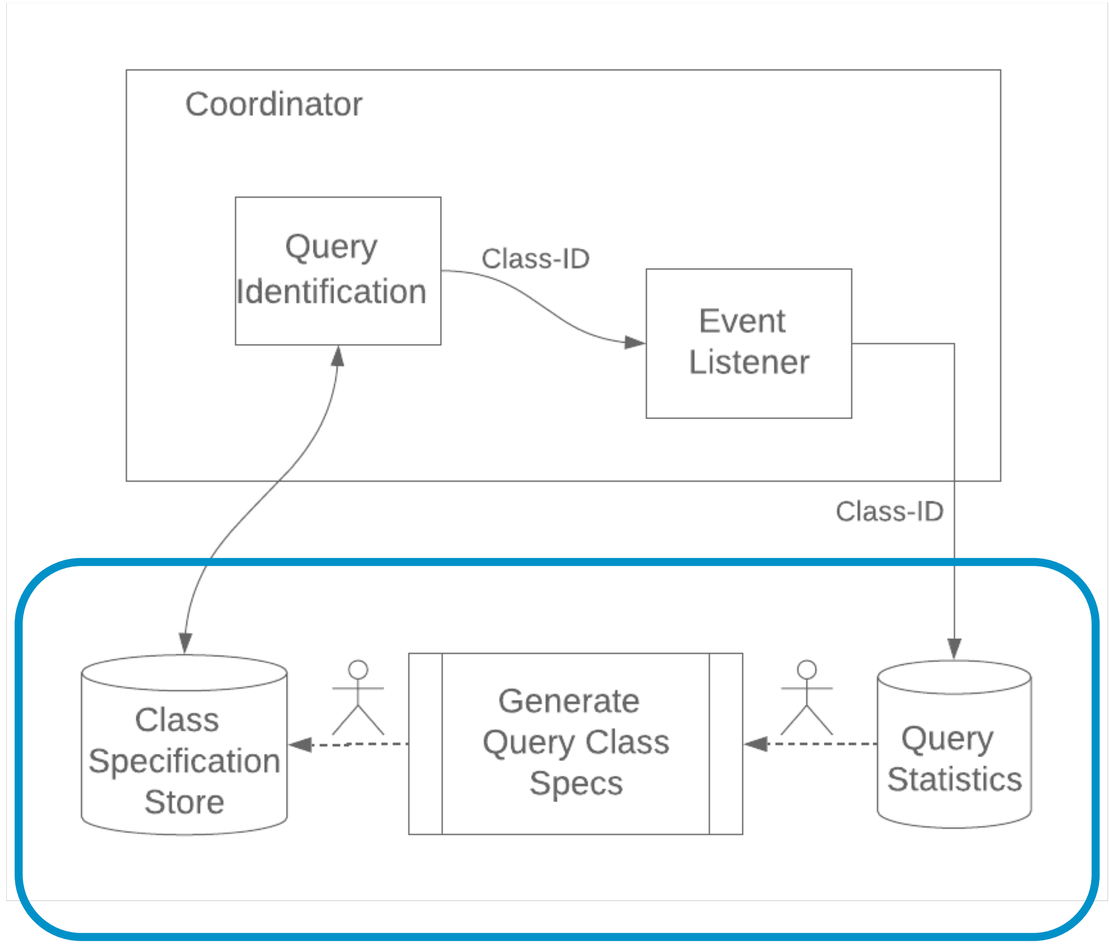
Class ID	Trait Name	Trait Value
classX	query_regex	'...shipmode = [:alpha:]'
classX	session_user	price_analysis
classX	dashboard_id	42
classY	query_regex	'...shipdate < date [:date:]..'
classY	session_user	price_analysis

Identification - performed by looking up traits and comparing against the query metadata

Query Identification - Notes

- Identification runs on coordinator
- Identified class IDs are emitted along with query lifecycle events
- Trait configurations can be stored externally (example MySql, Postgres, Configuration files)
- Currently a small number of simple traits are exposed
- Can be extended to more complex traits for example based on
  - Query Analysis
  - Intermediate Representation

Query Identification - Query Class Specification



Configuration - for new query classes the traits are determined through an offline procedure

The Value of Query Identification

Associating a unique ID with execution statistics of latency sensitive queries

Directly allows analytics on recurring queries including capacity use

Enables upfront discussions about latency needs of specific queries

Doing this before execution on coordinator allows query specific scheduling and planning changes

User Expectations

- Query identification allows negotiating an acceptable runtime for a query (SLA)
  - Can be done for any Query Class
  - Negotiated in advance
  - Used as a semi-formal agreement

- On-boarding an initial set of applications already shows the value of having these discussions with users upfront
- Deviations from SLA can be tracked
- Feeds into capacity planning discussions

Improving SLA Compliance

Several avenues are under consideration to increase the likelihood of SLAs being met for queries.

- **Scheduling Changes**
  - Efficiency vs Performance
  - Choosing different NodeSelectors based on class IDs
  - Static Isolation of nodes for running specific use cases
- **Session Properties**
  - Can use the SessionPropertiesConfigurationManager
  - Existing Framework
  - Provide ‘preferential’ treatment to certain query classes

Query Identification - Gotchas

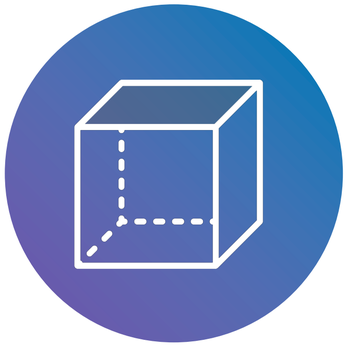
- Small differences in filter values can lead to significant differences in query execution, especially for complex queries
  - Worst case SLA may still be meaningful
  - Split into different classes if needed
- Some queries may be executed as part of an EXECUTE statement
  - Patch for capturing corresponding PREPARE statements is under review

https://github.com/prestodb/presto/pull/12020

Query Identification - Key Takeaways

Identification

Identify repeated executions of the ‘same’ query



Analytics

Analyze recurring queries

Scheduling

Online identification enables alternate scheduling

Flexibility

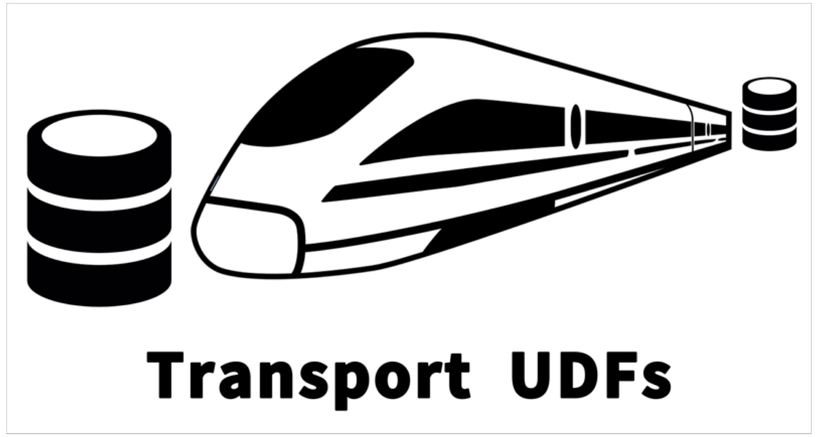
Toolkit for handling a rich variety of clients

Updates on Previous Work

- **Transportable UDFs**
- **Hive View Translation**

Slides from previous summit are located here

Transportable UDFs



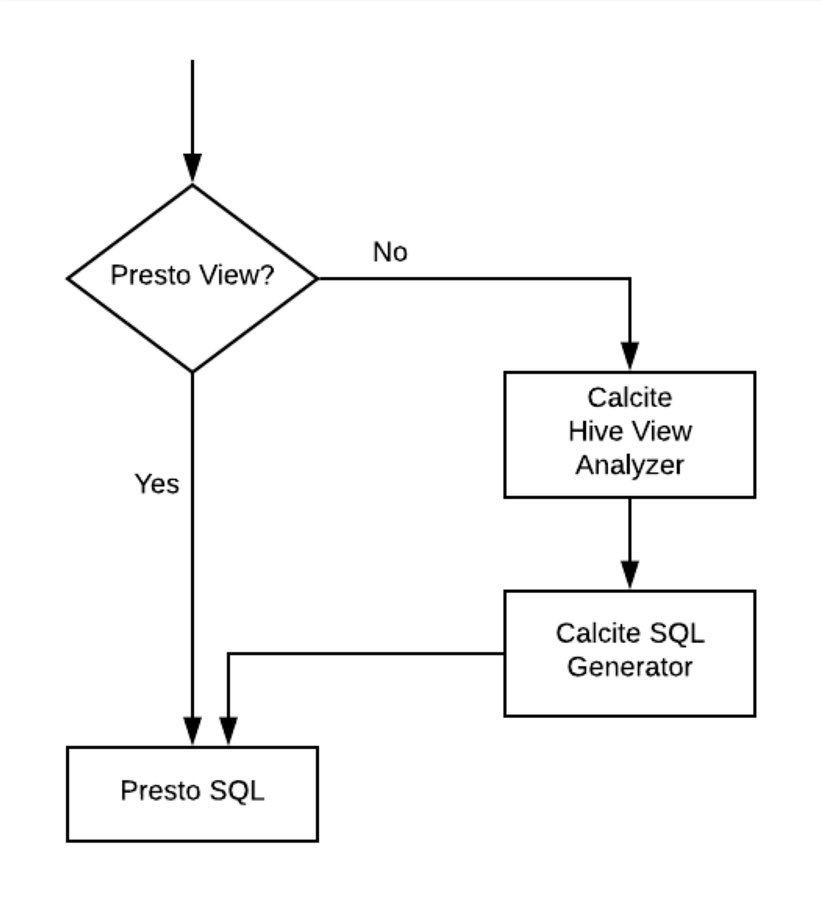
- Framework for writing UDFs that are portable across a variety of engines
- Supports Apache Spark, Apache Hive, and Presto
- Developed at LinkedIn and open sourced
- https://github.com/linkedin/transport

Transportable UDF Example

```
public class MapFromTwoArraysFunction extends StdUDF2<StdArray,
StdArray, StdMap> implements TopLevelStdUDF {

 private StdType _mapType;

 @Override
 public StdMap eval(StdArray a1, StdArray a2) {
   if (a1.size() != a2.size()) {
     return null;
   }
   StdMap map = getStdFactory().createMap(_mapType);
   for (int i = 0; i < a1.size(); i++) {
     map.put(a1.get(i), a2.get(i));
   }
   return map;
 }

 @Override
 public void init(StdFactory stdFactory) {
   super.init(stdFactory);
   _mapType =
getStdFactory().createStdType(getOutputParameterSignature());
 }.....
}
```

- Core function logic written only once
- Simple wrapper functions required for every engine - auto generation in progress

Hive Views to Presto SQL Translation



- Wealth of views written in Hive (Dali Views)
- Translate through Apache Calcite
- Coverage for most Hive features

Future Work

- **Presto on YARN**
  - Available capacity on YARN clusters
  - Improved resource utilization and elasticity
- **Partitioning**
  - Extend the current support for
    - Co-partitioning
    - Grouped Execution
- **Log Analytics**
  - Analysis of service logs
  - Experimenting with Presto


LinkedIn