

Case Study from Treasure Data - How do users run query on Trino?

Trino Japan Virtual Meetup
November 17, 2021

Toru Takahashi
Director, Customer Experience
Treasure Data

About Me



Toru Takahashi

- **Twitter @nora96o / Github: toru-takahashi**
- **Started as 1st Technical Support Engineer at Treasure Data**
- **Current: Director, Customer Experience**
- **Disclaimer**
 - **Treasure Data uses Trino 312/350 but keep using “Presto” as our term for backwards compatibility**

Today's Talk

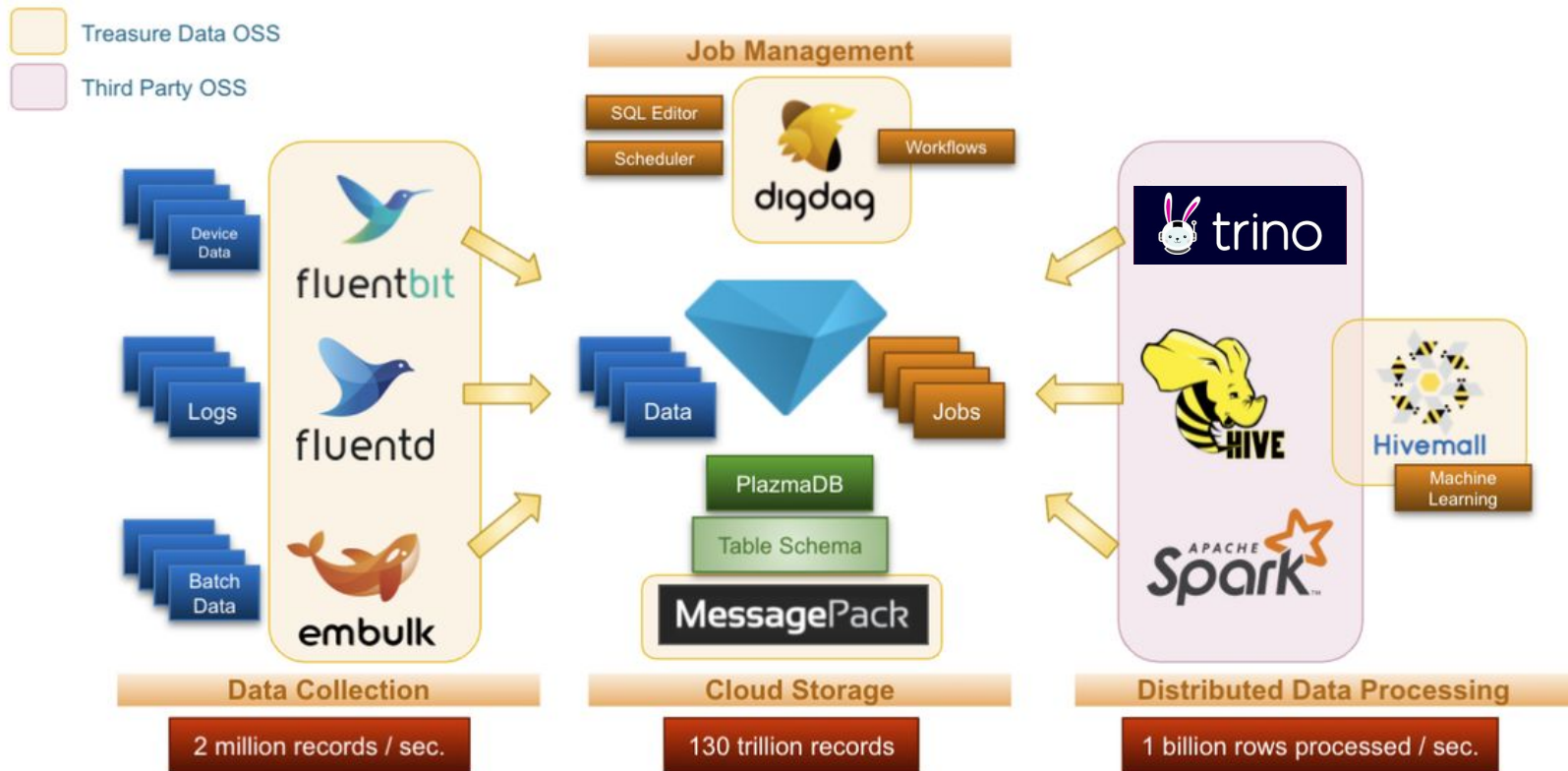
- **What I will talk today**

- Share a case study about our customers' Trino usage as use of customer data platform
- Give some insights if you start providing Trino to your internal/external end users

- **What I will not talk today**

- Details about Our implementation on our service with Trino
- Things about Trino configuration on our environment

Overall Architecture



How can users use Trino on TD?

Custom SQL

Segment Builder

The screenshot shows the Trino SQL editor interface. At the top, there's a header with "Queries/Untitled Query", "Draft", "Save", "Run", and a search icon. Below the header, there's a sidebar on the left with a "Search Tables" input and a list of 13 tables: campaign_performance_rollup, google_ads_campaign_performa..., keywords_performance_report, queries_20190304_1626, sample_report, search_query_performance_report, search_query_performance_repo..., segment_history_20190515, unnested_keyword_performance..., unnested_search_query_perform..., vikas_test, wish_queries_with_numerics_20..., and wish_queries_with_numerics_20... The main area displays a SQL query:

```
1 SELECT month, flag, count(  
2 FROM  
3 (  
4 SELECT 'eu01' as env, mo  
5 FROM  
6 ( SELECT CAST(td_time_  
7 month, account_id, max  
8 FROM production_eu01.e  
9 WHERE TD_INTERVAL(TIME  
10 AND (price_plan_id =  
11 OR presto_pla  
12 ) aws1  
13 CROSS JOIN UNNEST (try_c  
14 )) t1 (role)  
15 UNION ALL  
16 SELECT 'eu01' as env, mo  
17 FROM  
18 ( SELECT CAST(td_time_  
19 month, account_id, n  
20 feature_names  
21 FROM production_eu01.e  
22 WHERE TD_INTERVAL(TIME  
23 AND (price_plan_id =  
24 OR presto_pla  
25 ) aws2  
26 CROSS JOIN UNNEST (try_c  
27 (varchar))) t2 (featur  
28 )  
29 )  
30 )
```

The screenshot shows the Segment Builder interface. At the top, there's a header with "Segment 1", "jp_personalization_corporate+plazma Root", and a "Refresh" button. Below the header, there's a sidebar on the left with a "Search rules" input and a list of rules: referrer, mostrecententrytitle, and entrydescription. The main area displays the configuration for "Segment 1". It shows a "Refresh" button and a "Match logic" dropdown set to "all". Below this, there are two rules:

- Rule 1: Match logic "all". Rule A: td_country_name = is Japan. Rule B: privacypolicy = is yes.
- Rule 2: Match logic "all". Rule A: most_recent_utm_medium_c = is display.

The screenshot shows the SQL Code editor interface. At the top, there's a header with "Segment 1", "jp_personalization_corporate+plazma Root", and a "Refresh" button. Below the header, there's a sidebar on the left with a "Search rules" input and a list of rules: referrer, mostrecententrytitle, and entrydescription. The main area displays the SQL code for "Segment 1". It shows a "Refresh" button and a "Copy SQL Code" button. Below this, there's a text area containing the following SQL query:

```
1 select  
2 a.*  
3 from "cdp_audience_42828"."customers" a  
4 where (  
5 a."td_country_name" = 'Japan'  
6 and  
7 a."privacypolicy" = 'yes'  
8 )  
9 and (  
10 a."most_recent_utm_medium_c" = 'display'  
11 )  
12 )
```

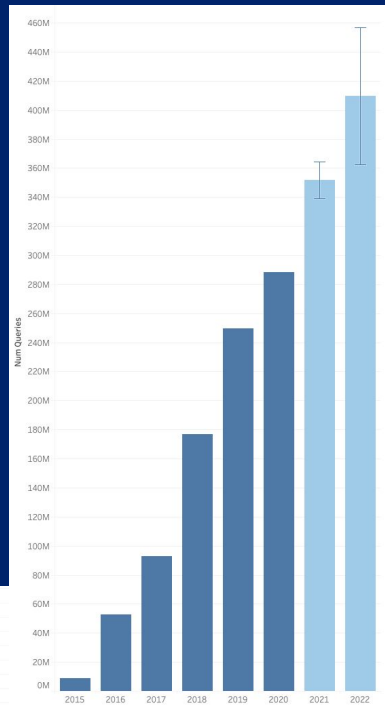
Trino Usage

- **Num of Queries**

- Executed 2.8 billion queries (num of unique query_id) in 2020
350 million queries (1.25x YoY) will be executed in 2021.
- Roughly 1 million queries per day

- **Num of users**

- Total Registered Users is 50K users





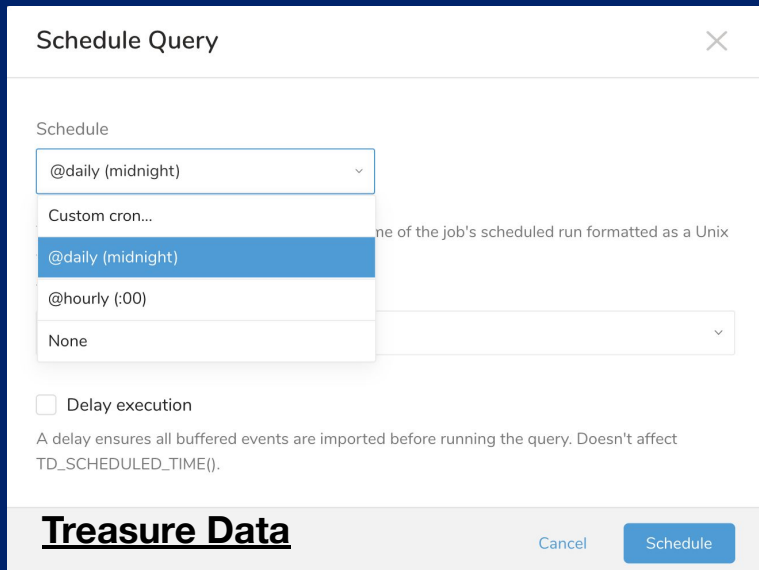
Let's see some interesting usages



How do users configure a schedule?

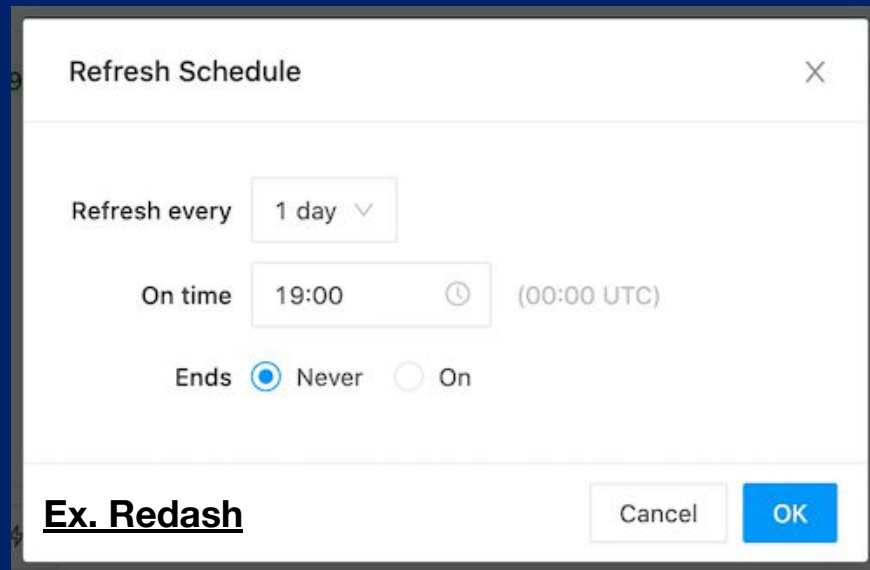
Scheduling Queries

Scheduled query to run your query automatically once a schedule is set is a common feature in a lot of BI / Data analytics Services.



The 'Schedule Query' dialog box features a 'Schedule' section with a dropdown menu currently showing '@daily (midnight)'. The dropdown list includes options: '@daily (midnight)', 'Custom cron...', '@daily (midnight)' (highlighted), '@hourly (:00)', and 'None'. Below the dropdown is a checkbox for 'Delay execution' with a descriptive note: 'A delay ensures all buffered events are imported before running the query. Doesn't affect TD_SCHEDULED_TIME().'. At the bottom, there are 'Cancel' and 'Schedule' buttons.

Treasure Data



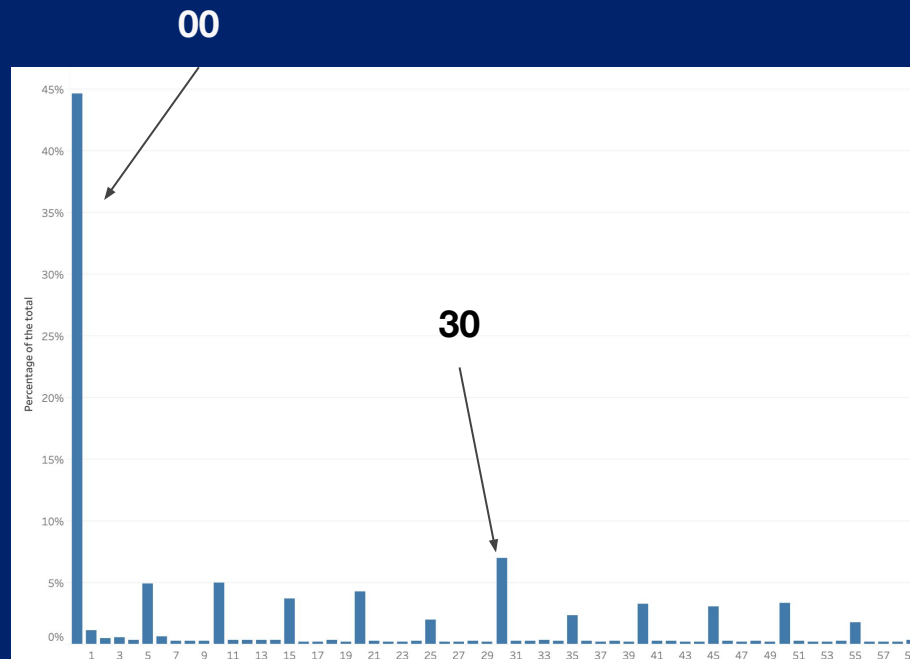
The 'Refresh Schedule' dialog box includes a 'Refresh every' dropdown set to '1 day'. Below it, the 'On time' field is set to '19:00' with a clock icon and '(00:00 UTC)' to its right. The 'Ends' section has two radio buttons: 'Never' (selected) and 'On'. At the bottom right, there are 'Cancel' and 'OK' buttons.

Ex. Redash

How do users configure a schedule?

The graph shows how people configure a query schedule...

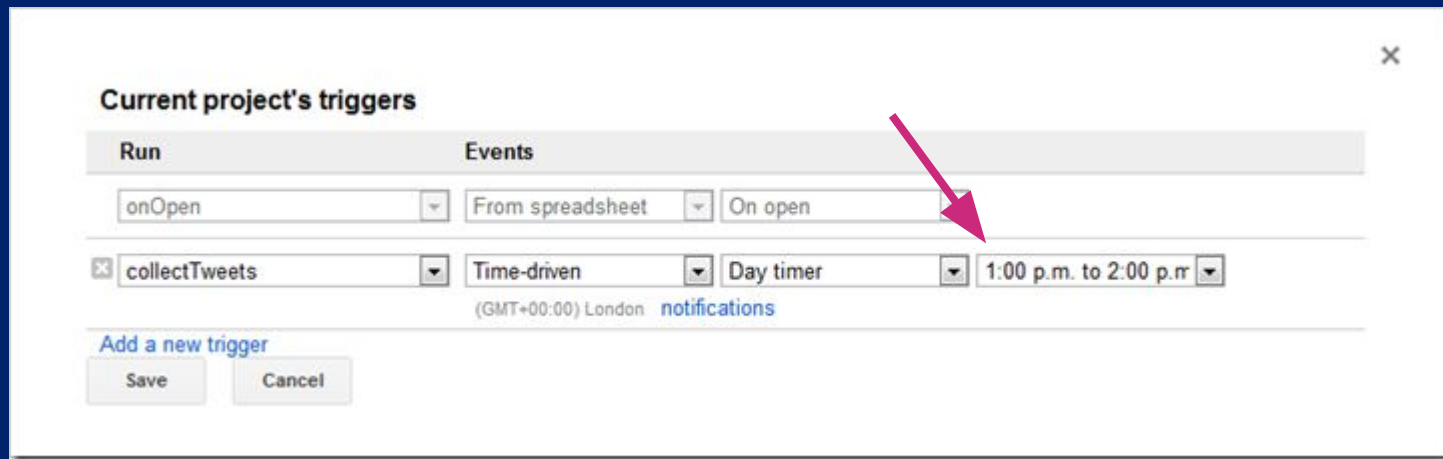
- 45% Scheduled Queries run on 00 minutes exactly
 - Users love Daily/Hourly/every 30 mins
- We had expected multi tenant system helps a distribution of job execution. However, Schedule is tightly associated with the data location
- Clusters always needs to adjust a daily peak workload because 1 hour is too short for auto-scaling



Side Note...

One Good Idea I have seen

- Google Sheet trigger function is a nice from system workload point of view. Users specify a certain period, not certain time.
- But, Generally speaking, providing this feature is a challenge...



Current project's triggers

Run	Events
onOpen	From spreadsheet On open
collectTweets	Time-driven Day timer 1:00 p.m. to 2:00 p.m.

(GMT+00:00) London notifications

[Add a new trigger](#)

Save Cancel

Query Execution Time

- 99% jobs completed with 10 mins however 0.005% queries (hundreds queries per day) run over 2 hours.
 - This long running jobs are critical for cluster stability
- At the beginning, we had 24 hours Query Execution Time restriction however it was too long. Thus, we have changed to 4 hours.
 - This makes us rotating a cluster hard. Because re-running long running jobs may take a double execution time by a retry

<1 min	94.5419%
1mins ~ 10mins	5.2535%
10mins - 60 mins	0.1860%
1 hour - 2 hours	0.0127%
2 hours - 3 hours	0.0027%
over 3 hours	0.0032%

SQL Text Length

- 96% of query length are less than 10,000 characters
- 0.1% (\leq hundreds) queries have over 1 million characters
= Text with over 1 mega bytes is sent to Trino ...
- Example...
 - WHERE col in ('x','y','z', ... million of IN condition)
 - UNION ALL .. UNION ALL .. Union ALL ... over 10000 times
- Having a gateway to block a job
by a query text size in front of Trino

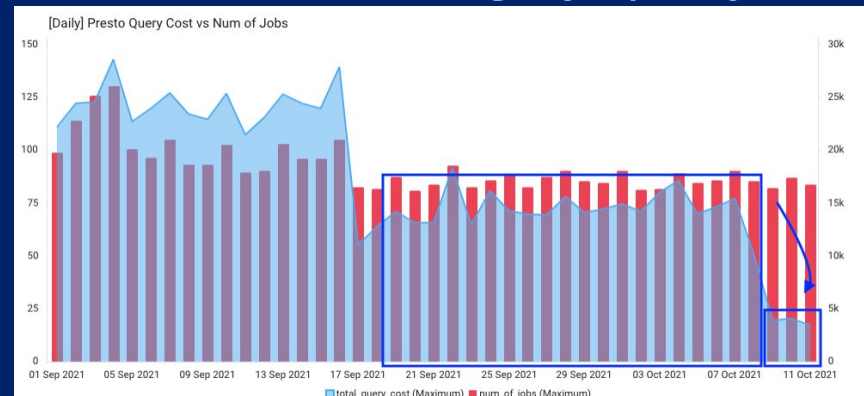
length<1000	74.23%
1000<=length<10000	22.79%
10000<=length<100000	2.80%
100000<=length<1000000	0.18%
length>=1000000	0.01%

Measuring Presto Query Cost in \$\$\$

- It's important to understand how much money we are spending for processing individual Trino queries even if our pricing model doesn't bill a query processing.
 - Calculated by the machine hour usage of individual queries (e.g., CPU, memory hour, splits executing time, network capacity occupancy hours), as well as more detailed stats, such as S3 GET count, processed rows and bytes, etc
- It motivate us to improve a customer query by education, suggestion, etc.

Re-Partitioning is a key of performance and Cost

- Our table requires to use a specific key for partitioning and they can't manage the partitioning info in detail. And, we can't regularly re-merge partitions due to a massive data volume
- Scanning customer fragmented S3 partitions was quite expensive in terms of the S3 GET cost.
- Remerging fragmented partitions based on finding by query cost data



S3 Access Stability Improvement

- Switched our S3 internal access method from the old path-style requests:
https://s3.amazonaws.com/(bucket_name)/
to the virtual-hosted S3 request style:
https://(bucket_name).s3.amazonaws.com/
 - According to AWS, it enables various internal optimization for managing the S3 traffic.
- This improvement has stabilized the table scan performance of Presto.
- Cost efficiency
 - S3 GET request cost: \$0.0004 / 1000 requests.
\$X/day*365d * multiple regions >= \$10K



Learn from Our Customers

- Users will do things beyond your worst expectations.
 - SQL is one of the popular and common tool to access data. But people don't know how to write efficient SQL yet.
 - But insufficient queries will be a huge pain from operation and cost point of views.
- In B2B, to strict a limit from current is so difficult than to relax a limit.
- My recommendation is to establish a strict limit at first, and then relaxing the limit gradually.

We are Hiring!

<https://jobs.lever.co/treasure-data>

In Japan, US, UK, Canada, +α