Even Faster:
When Presto Meets Parquet

@ Uber

Zhenxiao Luo
Software Engineer @ Uber
Mission
Uber Business Highlights
Analytics Infrastructure @ Uber
Presto
  Interactive SQL engine for Big Data
Parquet
  Columnar Storage for Big Data
Parquet Optimizations for Presto
Ongoing Work
Uber Mission

Transportation as reliable as running water, everywhere, for everyone
<table>
<thead>
<tr>
<th>Category</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Continents</td>
<td>6</td>
</tr>
<tr>
<td>Countries</td>
<td>73</td>
</tr>
<tr>
<td>Cities</td>
<td>450</td>
</tr>
<tr>
<td>Employees</td>
<td>12,000</td>
</tr>
<tr>
<td>Avg. Trips/Day</td>
<td>10+ Million</td>
</tr>
<tr>
<td>MAU Riders</td>
<td>40+ Million</td>
</tr>
<tr>
<td>MAU Drivers</td>
<td>1.5+ Million</td>
</tr>
</tbody>
</table>
Parquet @ Uber

Raw Tables
- No preprocessing
- Highly nested
- ~30 minutes ingestion latency
- Huge tables

Modeled Tables
- Preprocessing via Hive ETL
- Flattened
- ~12 hours ingestion latency
Scale of Presto @ Uber

- 2 clusters
  - Application cluster
    - Hundreds of machines
    - 100K queries per day
    - P90: 30s
  - Ad hoc cluster
    - Hundreds of machines
    - 20K queries per day
    - P90: 60s
- Access to both raw and model tables
  - XX petabytes of data
- Total 120K+ queries per day
Applications of Presto @ Uber

- Marketplace pricing
  - Real-time driver incentives
- Communication platform
  - Driver quality and action platform
  - Rider/driver cohorting
  - Ops, comms, & marketing
- Growth marketing
  - BI dashboard for growth marketing
- Data science
  - Exploratory analytics using notebooks
- Data quality
  - Freshness and quality check
- Ad hoc queries
What is Presto: Interactive SQL Engine for Big Data

Interactive query speeds

Horizontally scalable

ANSI SQL

Battle-tested by Facebook, Uber, & Netflix

Completely open source

Access to petabytes of data in the Hadoop data lake
How Presto Works

1. SQL Query
2. Abstract Syntax Tree
3. Logical Plan
4. Tasks
5. Scheduling
6. Membership & Status Tracking
7. Reading from HDFS
8. SQL Operations
9. Streaming Result

Client

Metastore

Coordinator

Analyzer

Fragmenter

Planner

Scheduler

Node Manager

Worker

Aggregation

Worker

Table Scan

Worker

Table Scan

Worker

Table Scan

Hadoop Distributed File System
Why Presto is Fast

- Data in memory during execution
- Pipelining and streaming
- Columnar storage & execution
- Bytecode generation
  - Inline virtual function calls
  - Inline constants
  - Rewrite inner loops
  - Rewrite type-specific branches
Resource Management

- Presto has its own resource manager
  - Not on YARN
  - Not on Mesos
- CPU Management
  - Priority queues
    - Short running queries higher priority
- Memory Management
  - Max memory per query per node
  - If query exceeds max memory limit, query fails
  - No OutOfMemory in Presto process
Limitations

- No fault tolerance
- Joins do not fit in memory
  - Query fails
  - No OutOfMemory in Presto process
    - Try it on Hive
- Coordinator is a single point of failure
Presto Connectors

From: www.mysql.com

From: www.mongodb.com

From: www.postgresql.org

From: kafka.apache.com

From: prestodb.io

From: hadoop.apache.com

From: cassandra.apache.com

From: redis.io
Parquet: Columnar Storage for Big Data
Parquet Optimizations for Presto

Example Query:

SELECT base.driver_uuid
FROM hdriveone.mezzanine_trips
WHERE datestr = '2017-03-02' AND base.city_id in (12)

Data:

- Up to 15 levels of Nesting
- Up to 80 fields inside each Struct
- Fields are added/deleted/updated inside Struct
Old Parquet Reader

Step 1: Read all Parquet nested fields from disk

Parquet

Step 2: Transform Parquet rows into Presto columnar blocks

Parquet

Step 3: Evaluate predicates on columnar blocks

Parquet
Nested Column Pruning

**Step 1: Only read required fields from disk**

**Step 2: Transform Parquet rows into Presto columnar blocks**

- base.driver_uuid base.city_id
- base.driver_uuid base.city_id
- base.driver_uuid base.city_id

**Step 3: Evaluate predicates on columnar blocks**

- base.driver_uuid base.city_id
- base.driver_uuid base.city_id
Columnar Reads

Step 1: Read required fields from disk
Build columnar blocks on the fly

Step 2: Evaluate predicates on columnar blocks

Parquet

Presto Columnar Engine
Dictionary Pushdown

Presto Columnar Engine

Read required fields from disk
Evaluate predicates on the fly
predicate: base.city_id = 12
row group city_id dictionary: {3, 5, 9, 14, 21}
skip reading row group
Build columnar blocks on the fly

Parquet

Row Group
Column Chunk base.client_uuid
Column Chunk base.driver_uuid
Column Chunk base.city_id
Column Chunk base.vehicle_id
Column Chunk base.status

Row Group
Column Chunk base.client_uuid
Column Chunk base.driver_uuid
Column Chunk base.city_id
Column Chunk base.vehicle_id
Column Chunk base.status

Parquet Footer: File Metadata, Row Group Metadata
Lazy Reads

Presto Columnar Engine

Read required fields from disk
Evaluate predicate on the fly
Build columnar blocks only if predicate matches

Parquet

Column Chunk
base.client_uuid

Column Chunk
base.driver_uuid

Row Group

Column Chunk
base.city_id

Column Chunk
base.vehicle_id

Column Chunk
base.status

Row Group

Column Chunk
base.client_uuid

Column Chunk
base.driver_uuid

Parquet Footer: File Metadata, Row Group Metadata
Benchmarking Results

Parquet Readers for Presto

- Old Parquet Reader
- New Parquet Reader

Execution Time in seconds

Benchmark Queries
Ongoing Work

- Multi-tenancy support
- High availability for coordinator
- Geospatial optimization
- Authentication & authorization
We are Hiring
https://www.uber.com/careers/list/27366/

Send resumes to:
abhik@uber.com or luoz@uber.com

Interested in learning more about Uber Eng?
Eng.uber.com
Follow us on Twitter:
@UberEng

Thank you