VeniceDB

a Peta-byte scale real time analytics service running Postgres on Azure

Min Wei

PGCon 2019

Agenda

- Use Case
- How Citus/Postgres is used
- Why Citus/Postgres
- Discussions

Use Case

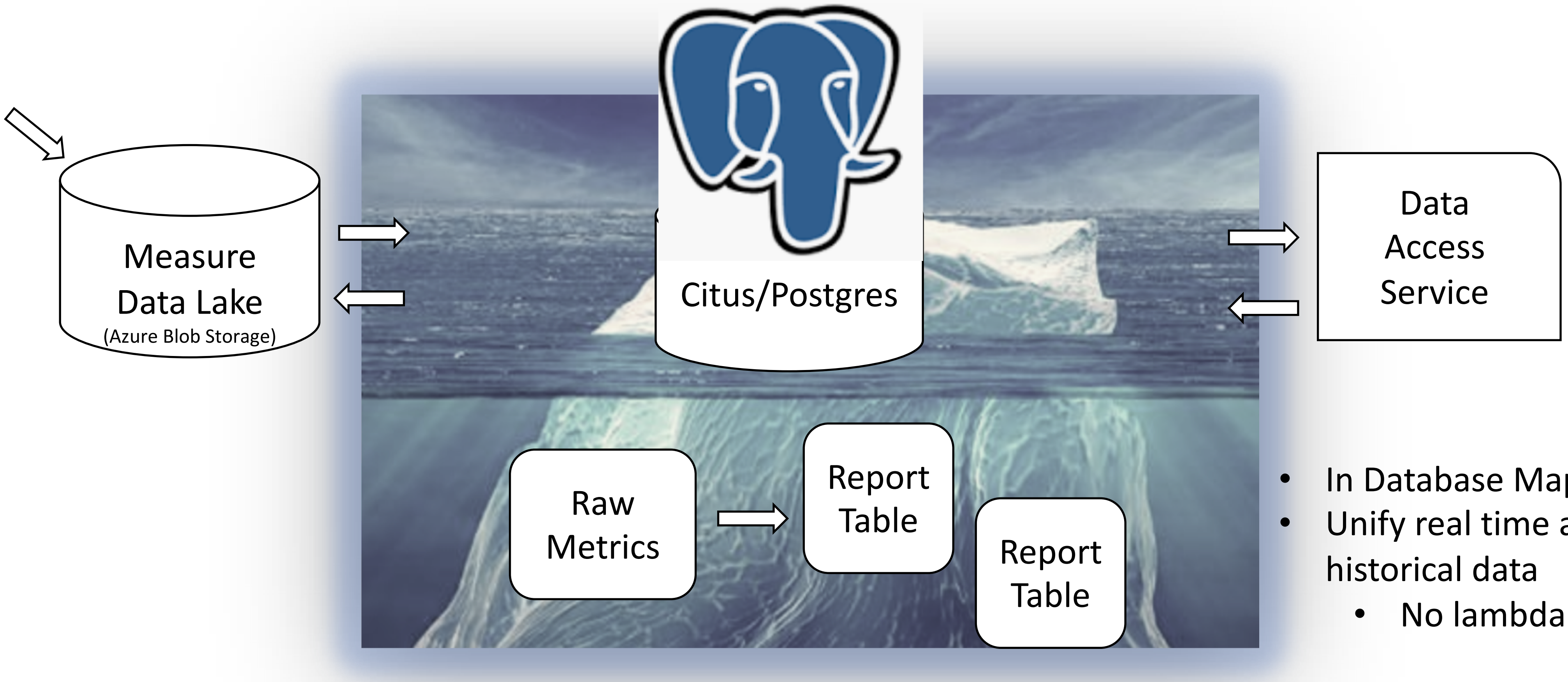
- Data warehouse for OS Measure Data
  - Covers over 800M unique devices monthly
  - Windows generates over 100K events, a few PBs/day
  - Measure is mostly computed from complex event joining
    - Measure user experience, reliability, …
      - E.g. Start Menu Click event + Start Menu Launch event => Start Menu launch latency
    - Dimension is enriched at server side batch processing
    - 5TBs per day
- Executive decision dashboard
  - 1000s of monthly active users
  - 100s of dashboard pages
  - 6+ million queries per day

* https://blogs.windows.com/windowsexperience/2019/03/06/data-insights-and-listening-to-improve-the-customer-experience/#j0PWI3cZMlHptKbC.97

Before VeniceDB

- There were solutions being developed and in production
  - Gen1 Sharded SQL row store
  - Gen2 Sharded SQL columnar store
    - Distributed columnar store
- Hard problems
  - Unable to scale to dimensionality
    - Requires MapReduce batch processing to prepare data cubes
      - Combinatorial growth
      - Data skew
  - Unable to deal with data variety and calculation complexity
    - Some SQL Scripts are over 500KB!
  - Unable to serve concurrent dashboard queries

VeniceDB Service Architecture



- In Database MapReduce
- Unify real time and historical data
  - No lambda!

Database Cluster Overview

- A custom build of Citus-Enterprise 8.2 and Postgres 11.3
    - Started from Citus 7.0 and Postgres 10.0
    - Leverage new features: parallel Btree indexing, parallel query, JIT, …
- Production database cluster
    - 2816 Cores, 18TB DRAM, 1PB Azure Premium Storage, Multi-PB Azure Blob Storage
        - 2 Physical clusters behind a query router (Azure Web Service and Azure Redis Service)
    - 20K+ measures over 800M unique monthly active devices
    - Ingest and delete ~5TB data per day
    - P75 query latency ~90ms/200ms
    - Support long running queries up to 4 mins.
    - Support batch scheduled jobs that can run up for 2hours

Measure Data Integration

- ~10 types of schema
- Unified schema

```
CREATE TABLE measures (
    measureid int,
    eventdatetime TIMESTAMPTZ,
    streamdatetime TIMESTAMPTZ
    data jsonb
    hashPartitionKey  bigint,
) PARTITION BY RANGE (streamdatetime);
```

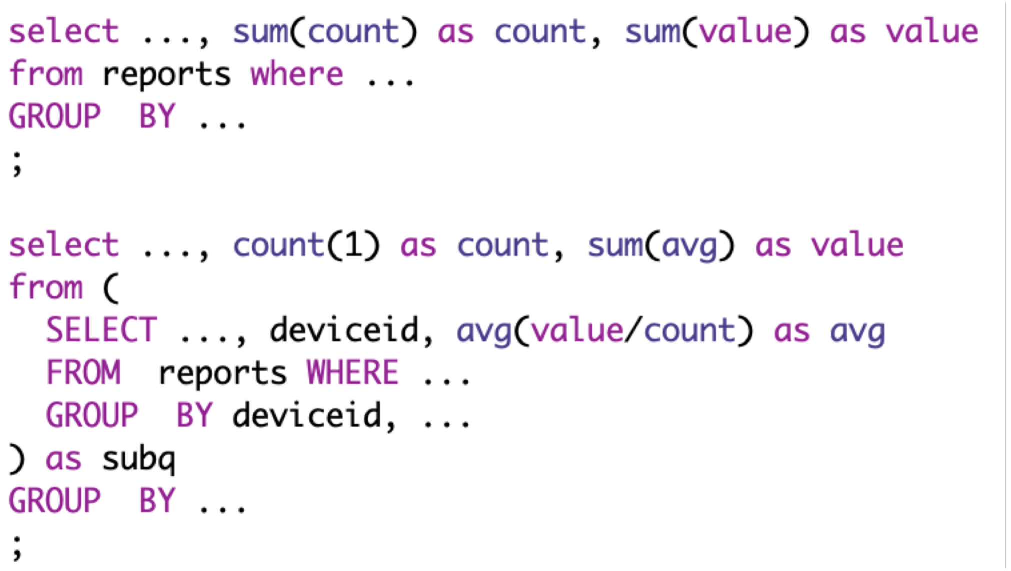
- Ingestion runtime
  - Most data arrives hourly, and business metrics data at minute time grain
  - Offsets table stored on the Citus coordinator node
  - Standalone go program running on Citus coordinator
    - Impressive GC in golang runtime, no visible pause with 100GB heap, run for months continuously

In Database MapReduce

- Slice the JSON data
  - Citus Distributed Upsert
    - Co-located with original measures data
    - Reshuffle through coordinator
  - Partition ~1B rows per hour into 15 day buckets
    - Report table time range partitioned by eventdatetime
    - Parallel upsert into 15 daily tables
    - Total # of jobs = 15 x ~10 kinds of report
  - JSON dimensions are kept for some tables
  - Heavily indexed daily tables
    - Over 50 partial covering indexes
    - Trade disk space for performance

Serving Queries

- Multi-dimensional calculations
  - On demand
  - Index Only Scan
- Data types
  - Metric
    - <count, sum>
    - {<weight, value>}
    - histograms
  - Dimension
    - Scalar
    - Array
- Aggregation types
  - Average
  - Percentile
  - Latest Value



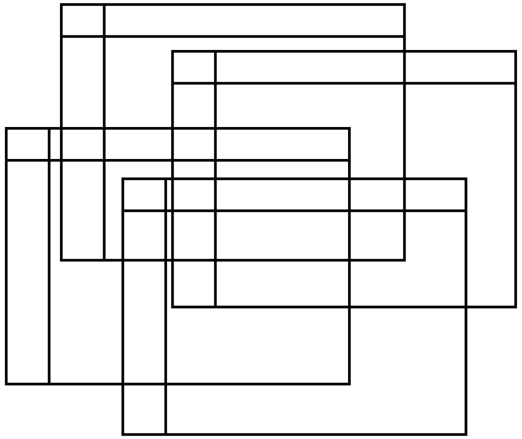
```
select ..., sum(count) as count, sum(value) as value
from reports where ...
GROUP  BY ...
;

select ..., count(1) as count, sum(avg) as value
from (
  SELECT ..., deviceid, avg(value/count) as avg
  FROM  reports WHERE ...
  GROUP  BY deviceid, ...
) as subq
GROUP  BY ...
;
```

What about alternatives?

- There are many choices!
  - Almost all SQL Columnar Store
  - Apache Projects: Kylin, Pinot, Druid, Spark (DataBrick)
- Issues
  - SQL engine cannot handle nested high cardinality group by queries
    - Lack of extensible data types such as array
  - Too much lean on one trick: scanning
    - Much less compression rate with JSON data, in particular GUID fields
  - Slow concurrent query performance to drive large scale decision dashboards
  - No updates when you really need them
    - Data can be wrong for various reasons, fixing data is often better than redeploying services!
  - Additional management complexity and data preparation services
    - Hive/Spark, HDFS/HBase, ZooKeeper

Discussions I

- Declarative Programming takes time to practice!
- Data structure
  - Table schema
  - Indexing
  - Partitioning and Paging
- Algorithms
  - Distributed query requires data movement aware
  - Postgres has rich data types and join types
- Execution
  - Master “Explain”
  - Locking
  - Watch auto vacuum thresholds
  - Grafana as a visual debugger
    - IO
    - Memory
    - Networking
    - Disk utilization
    - Connection count



Discussions II

- Faster partition pruning
  - ~3x performance degradation, 30 partitions vs 180 partitions
- Built-in Connection Pool Manager
  - selective turn off table access for maintenance besides connection pooling
- More fine grained control over shared_buffers
  - currently custom warmup code to load relative cold indexes
  - Compress cold index pages?
- Vectorized query execution
  - built-in column store will also help additional scenarios
- Better options for partition table migration between table spaces
  - Tiered table from hot data to cold data
- Stats function
  - Many won’t work in a distributed SQL environment