Towards Dynamic Data Placement for Polystore Ingestion

Jiang Du (University of Toronto)
John Meehan (Brown University)
Nesime Tatbul (Intel Labs & MIT)
Stan Zdonik (Brown University)
The IoT (Internet of Things) Era

- High data velocity
- High data variety
- High data volume
- High data veracity

Question: How to ingest data and make it available for analysis and control fast?
Traditional Data Ingestion (ETL)

- Heterogeneous types of data collected into flat files
- Manage correctness of data ingestion
- Incremental update from an OLTP source, often once per day
Motivations: IoT

- Data ingestion for IoT
  - Ingests data from *a variety of sources*
  - Guarantees *correctness of data ingestion*
  - Requires *low ingestion latency* for streams of data
  - Requires a *coherent way* to query the ingested data

- Data currency & low latency is key for fast decision making & control
  - Value of ingested data decreases over time
  - Data ingestion & data analysis executed on specialized engines
Polystores and Streaming Engines

- **Polystores (BigDAWG)**
  - “One-size-fits-all” no longer stands
  - Built on top of a variety of specialized databases
  - Provide **transactional data migrations** between engines
  - Provide a coherent query interface

- **Stateful streaming engines (S-Store)**
  - Ingest data from a variety of sources
  - Support **correct state management**
  - Scalable main-memory system with **low ingestion latency**
  - Support **hybrid streaming+OLTP+OLAP workloads**
Polystore Ingestion: A Simplified Architecture
Problem: Data Placement Plan

- Assuming
  - Hybrid ETL + OLTP + OLAP query workloads
  - Workload characteristics evolves
  - Tables are partitioned into data fragments

- Should a data fragment be stored in the streaming engine, in the OLAP engine, or both?
Data fragments can be migrated between database engines

- Two options during migration: **Copy** or **Move**

  - **Copy** (caching): Keep the fragment in its source engine
    - Incurs redundant storage and extra cost for maintaining consistency
    - Guarantees local data access

  - **Move** (anti-caching): Delete the fragment from its source engine
    - Keeps a single copy, avoids redundant storage and simplifies maintaining transactional consistency
    - No guarantee for local data access
Data Placement Plan: Challenges

- Different database engine characteristics
  - Streaming engine
    - Fast data access with limited storage capacity (memory-based)
    - Optimized for short-lived read and write transactions
  - OLAP engine
    - Slow short-lived transactions with large storage capacity (disk-based)
    - Optimized for read-intensive OLAP queries
- Evolving workload characteristics
- Data placement plan: must account for the characteristics of different database engines and the evolving workloads
Architecture: Data Ingestion

- **Data Migrator**
  - Supports 2PC for durability
  - Supports "Copy" and "Move" modes
  - Supports a variety of destination database engines

- **Data Catalog**
  - Maintains the location & size of each fragment of the tables
  - Maintains timestamps for each fragment access
  - Maintains the cost of each fragment access for each SP
  - Maintains the cost of migration of each fragment from one db to another
Architecture: Streaming Engine

- Provides local in-memory storage
- Temporary staging of ingested batches and intermediate derived data
- Caching copies of old data fragments for ETL lookups
- Serving as primary storage for data fragments for frequent updates

An extension: Fragment Selection module
- Checking/updating data catalog
- Migrating data between S-Store and other database engines
- S-Store cache selection
Takeaway Messages

- A case for **Copy**
  - When there is a large amount of local reads

- A case for **Move**
  - When there is a large amount of local writes

- Migration cost is expensive
  - For some scenarios, minimizing migration cost may be a good enough objective function
  - For other scenarios, we should consider the difference of execution costs for read and write in different database engines

- Data placement plan must be adjusted dynamically for the workload
Ongoing Work

- Building a cost model
  - Estimates a relative cost for various data placement plans
  - Accounts for the extreme expense for data migration
- Integrating with the BigDAWG query optimizer tightly
  - Uses the cost model to choose the best query plan
- Reducing the maintenance cost of the catalog
- Experimenting with real-life datasets and workloads