Polystore Systems for Complex Data Management

Kyle O'Brien

HPEC 2017

Massachusetts Institute of Technology
Outline and Tutorial Objectives

• Part 1: BigDAWG Details
  – Review basic polystore terminology
  – Common polystore motivations
  – BigDAWG API and query syntax
  – BigDAWG system components in detail

• Part 2: Hands-on example
  – Stand up a polystore cluster for Postgres, SciDB, and Accumulo databases using Docker
  – Explore data on each database
  – Execute polystore queries with BigDAWG
Polystore Terminology Review

• Island:
  – An abstraction of database engines having a similar data model and query language.
  – Example: Relational Island: Postgres and MySQL have tabular data structures and operate with SQL.

• Shim:
  – A query transformation operation written in one language but intended for a separate Island. Transparently handles the intended operation on the target system in terms of another language.
  – Example: SQL SELECT * FROM table is equivalent to SciDB SCAN(table).

• Cast:
  – A data transformation operation from one data model to another.
  – Example: integers can be cast to floats or to string representations.

BigDAWG Architecture

Visualizations

Clients

Applications

BigDAWG Common Interface/API

Relational Island

Array Island

Island ...

Relational DB

Array DB

… DB

Shim

Shim

Shim

Shim

Shim

Cast

Cast

Islands provide the intersection of engine capabilities. The BigDAWG Interface provides the union of Island capabilities.
### Analytical Optimization
- Data resides on one database engine but desired analytic operations are slow.
- We would like to select data from one database and analyze it on another database.

### Location Transparency
- Data is dispersed among several database engines.
- We would like to transparently query for it without having to know the specifics of what data is on what system.

### Organizational Legacy
- Data is already distributed among many databases and/or engines because policy or other legacy reasons.
- We would like to query across them while keeping data intact.
Common Polystore Motivations

Analytical Optimization
- Data resides on one database engine but desired analytic operations are slow.
- We would like to select data from one database and analyze it on another database.

Example:

- Data exists in Postgres, but desired analytical operations are slow.
- Analytical operations in SciDB are fast.

```
SELECT()  CAST()  FILTER()
Postgres  SciDB
```

Required Steps:
1. Select data in Postgres
2. Cast data to SciDB
3. Perform filtering in SciDB
Currently, there is support for 3 Islands, each with its own functional token:

- `bdrel()` – the query targets the relational island and uses SQL.
- `bdarray()` – the query targets the array island and uses SciDB’s AFL query language.
- `bdtext()` – the query targets the text island and uses either SQL or D4M.

Data can be explicitly cast from one Island to another with `bdcast()`:

```
bdcast( BIGDAWG_RETRIEVAL_SYNTAX,
        name_of_intermediate_result, {
            {, POSTGRES_SCHEMA_DEFINITION, relational}
            | {, SCIDB_SCHEMA_DEFINITION, array}
            | {, TEXT_SCHEMA_DEFINITION, text}} )
```
Example Polystore Query

bdarray(
  filter(
    bdcast(
      bdrel(
        SELECT val FROM table
      ),
      postgres_results,
      '<val:double> [i=0:*,1000,0]',
      array
    ),
    val < 35
  )
)
Example Polystore Query

```
bdarray(
  filter(
    bdcast(
      bdrel(
        SELECT val FROM table
      ),
      postgres_results,
      '<val:double> [i=0:*,1000,0]',
      array
    ),
    val < 35
  )
)
```

Step 1: Select data from RELATIONAL Island
Example Polystore Query

Step 2: Cast relational data to ARRAY Island

```
bdarray(
    filter(
        bdcast(
            bdrel(
                SELECT val FROM table
            ),
            postgres_results,
            '<val:double> [i=0:*,1000,0]',
            array
        ),
        val < 35
    )
)
```

# Intermediate result name
# Destination schema
# Cast destination island
Example Polystore Query

Step 3: Filter data within the ARRAY Island

```
bdarray(
  filter(
    bdcast(
      bdrel(
        SELECT val FROM table
      ),
      postgres_results,
      '<val:double> [i=0:*,1000,0]',
      array
    ),
    val < 35
  )
)
```
Query Endpoint

• The Query Endpoint accepts queries from clients over a network and returns results from the Middleware.

• The Query Endpoint is a simple HTTP server that accepts POST requests

Example Usage:

Purpose: BigDAWG must be aware of all database and data object components before it can handle queries. This information is all stored in the Catalog.

Implementation: The Catalog itself is a Postgres cluster with two databases: bigdawg_catalog and bigdawg_schemas.

The bigdawg_catalog database contains 5 tables:
- catalog.engines
- catalog.databases
- catalog.objects
- catalog.shims
- catalog.casts

The bigdawg_schemas database contains one table per data object. Each data object’s schema is encoded with a Postgres table schema.
# Catalog: Engines, Databases, and Objects

**catalog.engines**
- Engine connection properties.

**catalog.databases**
- Database-to-engine mapping.
- Database connection properties.

**catalog.objects**
- Object-to-database mapping
- Object field names.

### Engine Connection Properties

<table>
<thead>
<tr>
<th>eid</th>
<th>name</th>
<th>host</th>
<th>port</th>
<th>connection_properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>postgres0</td>
<td>bigdawg-postgres-catalog</td>
<td>5408</td>
<td>PostgreSQL 9.4.5</td>
</tr>
<tr>
<td>1</td>
<td>postgres1</td>
<td>bigdawg-postgres-data1</td>
<td>5401</td>
<td>PostgreSQL 9.4.5</td>
</tr>
<tr>
<td>2</td>
<td>postgres2</td>
<td>bigdawg-postgres-data2</td>
<td>5402</td>
<td>PostgreSQL 9.4.5</td>
</tr>
<tr>
<td>3</td>
<td>scidb_local</td>
<td>bigdawg-scidb-data</td>
<td>1239</td>
<td>Scidb 14.12</td>
</tr>
<tr>
<td>4</td>
<td>zookeeper.docker.local</td>
<td>zookeeper.docker.local</td>
<td>2181</td>
<td>Accumulo 1.6</td>
</tr>
</tbody>
</table>

### Database-To-Engine Mapping

<table>
<thead>
<tr>
<th>dbid</th>
<th>engine_id</th>
<th>name</th>
<th>user_id</th>
<th>password</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>bigdawg_catalog</td>
<td>postgres</td>
<td>test</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>bigdawg_schemas</td>
<td>postgres</td>
<td>test</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>mimic2</td>
<td>postgres</td>
<td>test</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>mimic2_copy</td>
<td>postgres</td>
<td>test</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>tpcch</td>
<td>postgres</td>
<td>test</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>tpcch</td>
<td>postgres</td>
<td>test</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>scidb_local</td>
<td>scidb</td>
<td>scidb123</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>accumulo</td>
<td>bigdawg</td>
<td>bigdawg</td>
</tr>
</tbody>
</table>

### Object Field Names

<table>
<thead>
<tr>
<th>objid</th>
<th>name</th>
<th>fields</th>
<th>logical_db_serial</th>
<th>physical_db_serial</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>mimic2v26.a_chartdurations</td>
<td>subject_id, icustay_id, itemid</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>mimic2v26.a_iodurations</td>
<td>subject_id, icustay_id, itemid</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>mimic2v26.a_meddurations</td>
<td>subject_id, icustay_id, itemid</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>mimic2v26.additives</td>
<td>subject_id, icustay_id, itemid</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>
• Purpose: Casting an object from one database to another requires knowledge of that object's schema.

• Object schemas are stored in the `bigdawg_schemas` database.

• Example:

**Data object: MIMIC II Admissions table**

<table>
<thead>
<tr>
<th>hadm_id</th>
<th>subject_id</th>
<th>admit_dt</th>
<th>disch_dt</th>
</tr>
</thead>
<tbody>
<tr>
<td>1459</td>
<td>83</td>
<td>2024-03-21 00:00:00</td>
<td>2024-03-28 00:00:00</td>
</tr>
<tr>
<td>2075</td>
<td>3</td>
<td>2024-09-07 00:00:00</td>
<td>2024-09-18 00:00:00</td>
</tr>
<tr>
<td>5712</td>
<td>61</td>
<td>2023-01-10 00:00:00</td>
<td>2023-02-09 00:00:00</td>
</tr>
<tr>
<td>7149</td>
<td>61</td>
<td>2023-06-23 00:00:00</td>
<td>2023-07-26 00:00:00</td>
</tr>
<tr>
<td>8743</td>
<td>94</td>
<td>2026-08-18 00:00:00</td>
<td>2026-09-10 00:00:00</td>
</tr>
</tbody>
</table>

**BigDAWG schema:**

```sql
CREATE TABLE mimic2v26.admissions
(
    hadm_id integer,
    subject_id integer,
    admit_dt timestamp without time zone,
    disch_dt timestamp without time zone
);
```

Postgres table definitions are used to encode all object schemas.
The Administrative Interface is a browser-based UI to view the Catalog and start/stop the cluster.
The Administrative Interface is a browser-based UI to view the Catalog and start/stop the cluster.

**BigDAWG Cluster Overview**

- **Query Endpoint (HTTP Server)**
- **Catalog**
- **Middleware** (Planner, Executor, Migrator, and Monitor)
- **PostgreSQL**
- **SciDB**
- **Accumulo**

**Catalog Viewer**

- **Engines**
  - Engine ID: 0, Name: postgres1, Host: bigdawg-postgres-catalog, Port: 5401, Connection Properties: PostgreSQL 9.4.5
  - Engine ID: 1, Name: postgres2, Host: bigdawg-postgres-data1, Port: 5402, Connection Properties: PostgreSQL 9.4.5
  - Engine ID: 2, Name: postgres3, Host: bigdawg-postgres-data2, Port: 5403, Connection Properties: PostgreSQL 9.4.5
  - Engine ID: 4, Name: zookeeper, Host: zookeeper_docker_local, Port: 2181, Connection Properties: Accumulo 1.6

- **Data Objects**
  - Object ID: 0, Table Name: mimic2x26_abbreviations, Contents / Schema: subject_id, icustay_id, ic_time, admission_id, duration
  - Object ID: 1, Table Name: mimic2x26_abbreviations, Contents / Schema: subject_id, icustay_id, ic_time, admission_id, duration
  - Object ID: 2, Table Name: mimic2x26_abbreviations, Contents / Schema: subject_id, icustay_id, ic_time, admission_id, duration
  - Object ID: 3, Table Name: mimic2x26_abbreviations, Contents / Schema: subject_id, icustay_id, ic_time, admission_id, duration
  - Object ID: 4, Table Name: mimic2x26_abbreviations, Contents / Schema: hadm_id, subject_id, admittime, discharge_time
The Planner coordinates all query execution:
- Query parsing, planning, and optimization
- Invokes the Executor with a final query execution plan to obtain results
The Planner works in two modes:

- **Training Mode (pre-production learning):**
  - Enumerate all orderings of execution steps that result in identical results (semantic equivalence).
  - Send plans to Monitor to gather execution metrics
  - Choose fastest plan
- **Non-Training Mode (production-ready):**
  - Extract features from the query and consult the Monitor for the best execution plan

---

The Executor performs query execution plans (QEPs) from either the Planner or Monitor.

The Executor traverses a QEP and issues sub-queries to databases for intra-island tasks and invokes the Migrator when data movement is necessary.

Performance information is sent back to the Monitor in training mode.
The Migrator abstractly handles data movement between two databases.

- Called by the Executor when needed to complete a query plan.
- Consults the Catalog to infer inter-island cast parameters.

Monitor

• The Monitor tracks the execution times of query plans as they are performed by the Executor.

• The Monitor runs training executions to learn execution metrics, and extracts query signatures from a query plan to infer optimal execution plans for new queries.

Middleware Components

MIMIC II dataset*
- Full dataset: over 3 terabytes (TB) total
- 1000s of intensive care unit patients from 2001-2008

Prototype BigDAWG Overhead

Overhead Incurred When Using BigDAWG
For Common Database Queries

- Count (Postgres)
- Average (Postgres)
- Average (SciDB)
- Standard Deviation (SciDB)
- Count (SciDB)
- Distinct Values (SciDB)

- Overhead Incurred (ms)
- Query without BigDAWG (ms)

Minimal Overhead

Medical Applications: HPEC'15
Goal: Find patients with similar ECG time-series*

Procedure
- Perform Discrete Wavelet Transform of ECG
- Generate wavelet coefficient histogram
- TF-IDF waveform coefficients (weight rare changes higher)
- Correlate against all other ECGs

Show timings for individual pieces in two different types of databases
- Option 1: Pick a DB and do the whole thing in it
  - This takes time and may not be ideal
- Option 2: Let the best DB for each piece do their part
  - Tough without some coordinator
  - Incur inter-database cast operation overhead

* A novel method for the efficient retrieval of similar multiparameter physiologic time series using wavelet-based symbolic representations, Saeed & Mark, AMIA 2006
Polystore Analytic Performance (1)

Time taken to perform analytic using different technologies

Technology Used
- SciDB
- Myria
- Hybrid

Time Taken (Seconds)

Better
Worse

Discrete Wavelet Transform
Term Frequency-Inverse Document Frequency
K-Nearest Neighbors

Medical Applications: HPEC’15
bdarray(  # Array island scope
  filter(  # Array island query operation
    bdcast(  # CAST operation
      bdrel(  # Relational island scope
        SELECT val  # Relational island query
        ),
        postgres_results,  # Intermediate cast result name
        '<val:double> [i=0:*,1000,0]',  # Destination schema
        array  # Cast destination island
        ),
        val < 35  # Finish array island query
      )
    )
  )
)
RELATIONAL(
    SELECT *
    FROM R, CAST(A, relation)
    WHERE R.v = A.v
)

This table lists all data objects managed by BigDAWG.

### catalog.objects

<table>
<thead>
<tr>
<th>oid [PK] serial</th>
<th>name character varying(50)</th>
<th>fields character varying(800)</th>
<th>logical_db serial</th>
<th>physical_db serial</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>mimic2v26.a_chartdurations</td>
<td>subject_id, icustay_id, itemid, 2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>mimic2v26.a_iodurations</td>
<td>subject_id, icustay_id, itemid, 2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>mimic2v26.a_meddurations</td>
<td>subject_id, icustay_id, itemid, 2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>mimic2v26.additives</td>
<td>subject_id, icustay_id, itemid, 2</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

- **Columns:**
  - **oid:** Object ID. The primary key.
  - **name:** Name of the data object (i.e., table name)
  - **fields:** comma-separated list of field names in the data object
  - **logical_db:** references catalog.databases.dbid, the original database where the object was created.
  - **physical_db:** references catalog.databases.dbid, the current location of the data object.

### logical_db and physical_db:

In case an object was created on one database and migrated to another, we track the initial "logical" database ID and the current "physical" database ID to know the original schema.
• Columns:
  – **shim_id**: primary key
  – **island_id**: ID for island
  – **engine_id**: references catalog.databases.eid to indicate the engine
  – **access_method**: To account for any version-specific language syntax (not currently used).