PolyBench: The first benchmark for polystores

Jeyhun Karimov, Tilmann Rabl, and Volker Markl

Polystore Background

- The term polystore was adopted from BigDawg paper
  - Many other terms, such as multi-store, multi-engine
  - BigDawg consists of Accumulo, SciDB, Postgres, S-Store
- Union of multiple specialized stores
  - Distinct language and execution semantics
  - Support for wide range of data types and analytics
- General-purpose single-store
  - Batch, Streaming workloads
  - ML, Graph computation use-cases
- Single-store vs polystore

Abstract

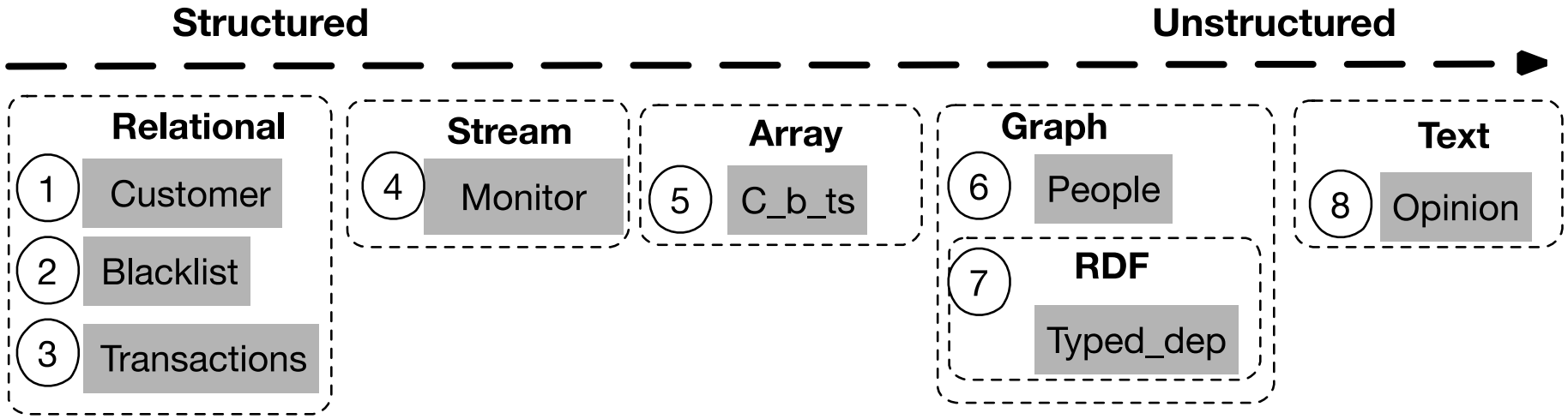
- **Goal:** Initialize the first benchmark for
  - Polystores
  - General-purpose single-stores
- Polystore papers:
  - Polystores overcome performance bottlenecks of single general-purpose stores
  - Polystores perform orders of magnitude better than single stores
  - The main evaluation method is TPC queries
- **Problem:** No polystore-specific metrics ,use-cases, and benchmarks
- Formalization of performance characteristics
  - “An important step to solve this problem is to find minimal set of evaluation use cases” – Jennie Duggan, BigDawg paper coauthor

Metrics and test scenarios

- Metrics
  - Runtime
  - Individual runtime
  - Idle time
  - Load

- Test scenarios
  - One-shot scenarios
  - Continuous scenarios
  - Resource distribution - nodes in a cluster, memory, and CPU
  - Load distribution - input data size and assigned subqueries for each member-store

Data model



- Simulation of banking bussiness model
- Unstructured, semi-structured, and structured data

Use-cases

- **Bank multi-model data integration**
- Combine relevant data in Graph-store
- Dependent use-case

```
INSERT INTO typed_dep VALUES (
CONVERT_INTO_RDF (
 SELECT *
 FROM Customer c
 WHERE c.updated > arg as u)
 UNION
 CONVERT_INTO_RDF (
 SELECT *
 FROM People p
 WHERE p IN u)
 UNION
 CONVERT_INTO_RDF (
 SELECT opinion_text
 FROM Opinion o
 WHERE o.ts > arg )
)
```

- **Customer background check**
- Unemployed, rich, alone
- Independent use-case

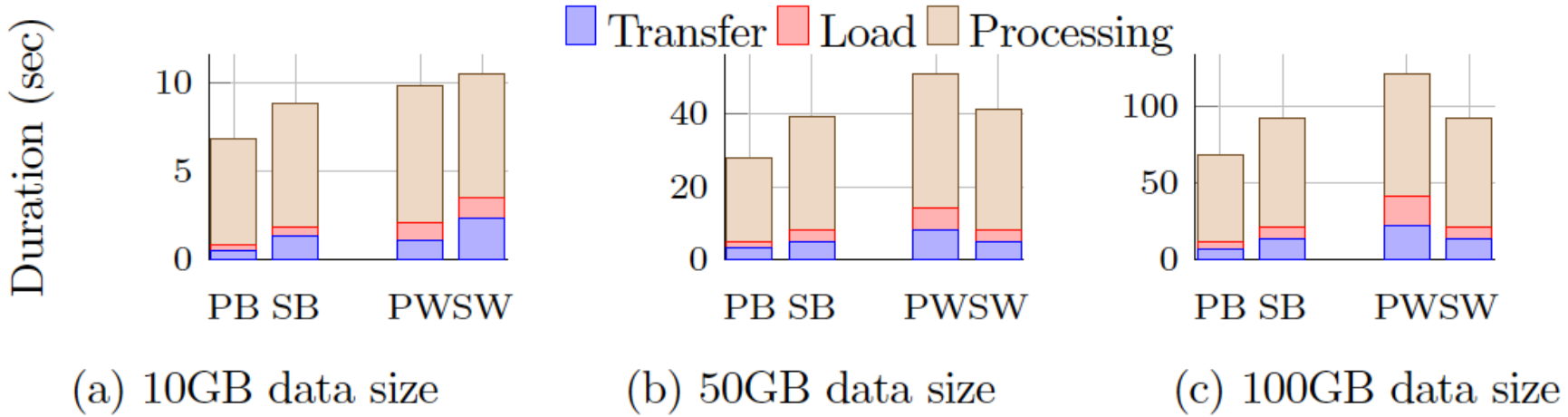
```
SELECT *
FROM (
 SELECT customer.userID
 FROM customer
 WHERE cutomer.work = null) AS c
(SELECT userID
 FROM c_b_ts
 WHERE c_b_ts.balance > arg1
 AND c_b_ts.year=arg2) AS c2,
(SELECT p.userID
 FROM people p
 WHERE p sp blacklisted < arg3)
 AS p

WHERE p.userID = c.userID
AND c.userID=c2.userID
```

Use-cases

- **Continuous queries: fraud detection**
- Enrich real-time data with relevant information from other member-stores
- If there is blacklisted ID, retrieve all transactions and balance information
- Dependent use-case

```
SELECT *
FROM customer c, transactions t, c_b_ts,
 (SELECT *
FROM monitor m
WHERE m.userID IN blacklist.userID)
 as fraud
WHERE c.userID = fraud.userID
AND t.userID = fraud.userID
AND fraud.userID = c_b_ts.userID
AND c_b_ts.ts within param_time
```

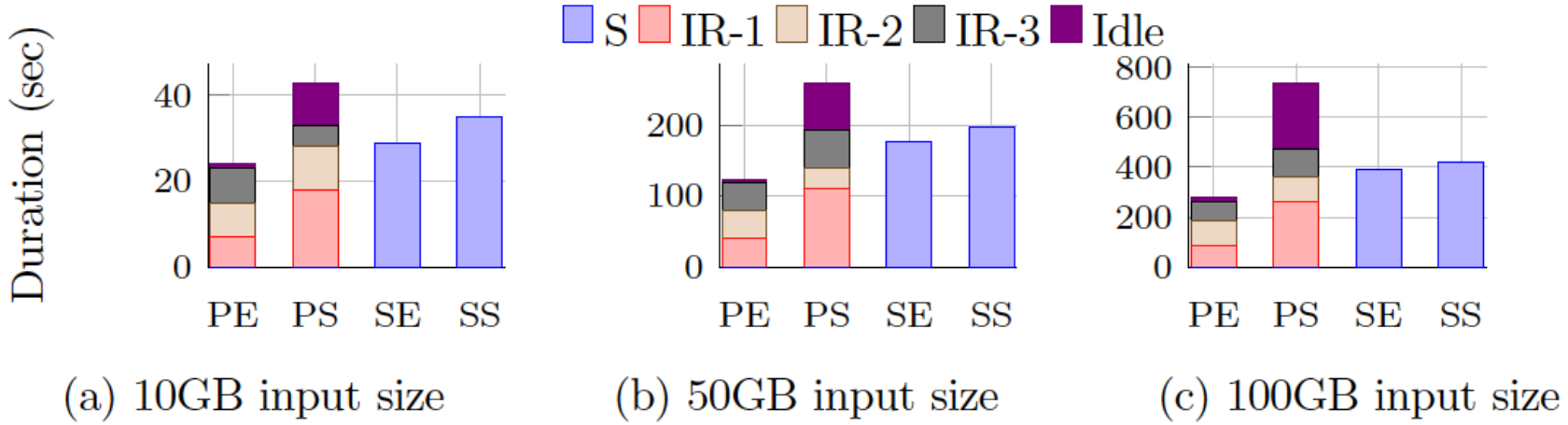
Experiments



(a) 10GB data size (b) 50GB data size (c) 100GB data size

- Use-case 1, Bank multi-model data integration
  - Effect of input data size for each member-store
- Resource utilization
- Relay-store (store-in-the-middle) bottleneck

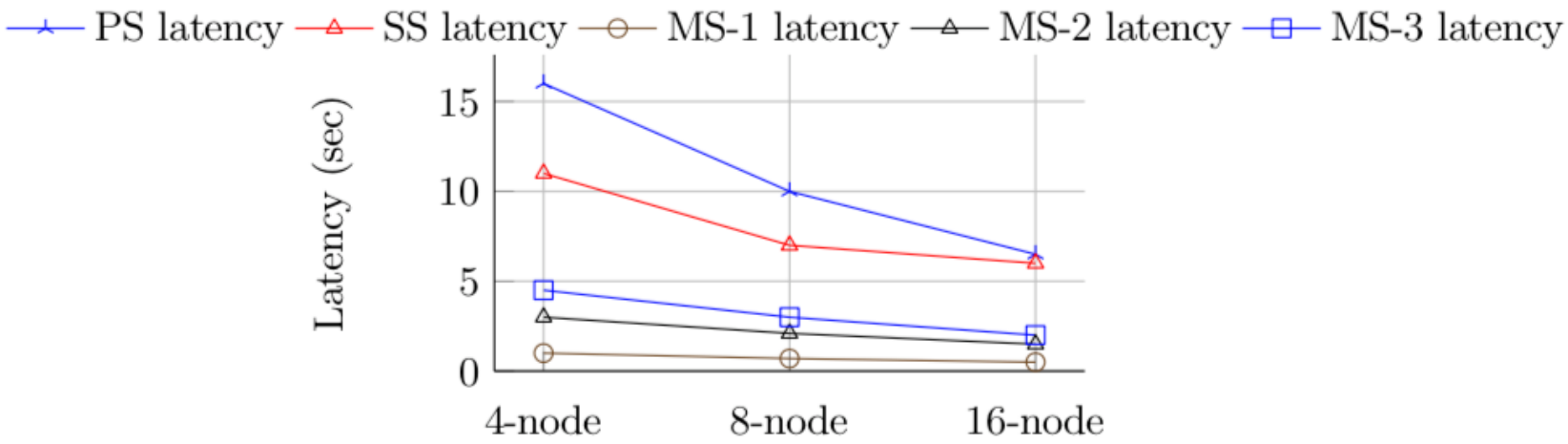
Use-case 2: Customer background check



(a) 10GB input size (b) 50GB input size (c) 100GB input size

- Effect of member-store load
- No blocking upstream stores
- Skewed vs shared load

Use-case 3:



- Continuous queries on polystores
- Input/output semantics of member-stores affect the performance

Conclusion

- This work initiates the first benchmark for polystores
  - Use-cases
  - Metrics
  - Test scenarios
  - Extensive experiments
- Need for a better
  - Polystore optimizer
  - Two-level scheduler
  - Data transfer layer
- Possible extension with graph/ML workloads