Towards a Modular Data Management System Framework

Haralampos Gavriilidis, **Lennart Behme**, Sokratis Papadopoulos, Stefano Bortoli, Jorge-Arnulfo Quiané-Ruiz, Volker Markl

September 9, 2022

A Brief History of Data Management Systems

- DMSes historically have a monolithic architecture
- Many special-purpose DMSes emerged since “one size does not fit all”
- The general architecture for query processors has not changed much since the 1980s though

⇒ Engineers reinvent the wheel as they have to build all components from scratch for each system

⇒ DMS components should be composable

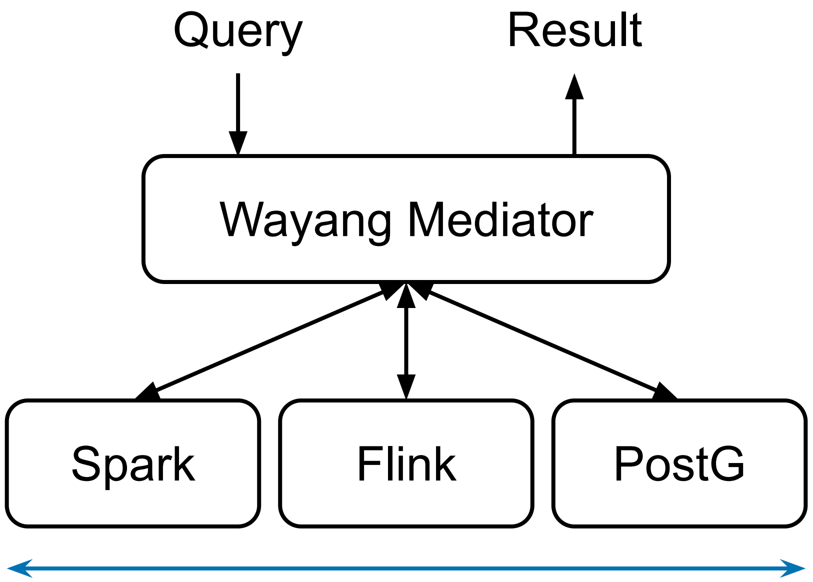
The State of the (Composability) Union

- There are no standards that allow system engineers to make their components easily composable
- Composing existing components into a DMS requires a huge amount of manual effort
- Approaches like Presto are impractical and costly as they continuously port data and pipeline logic from legacy DMSes to the latest and most advanced system

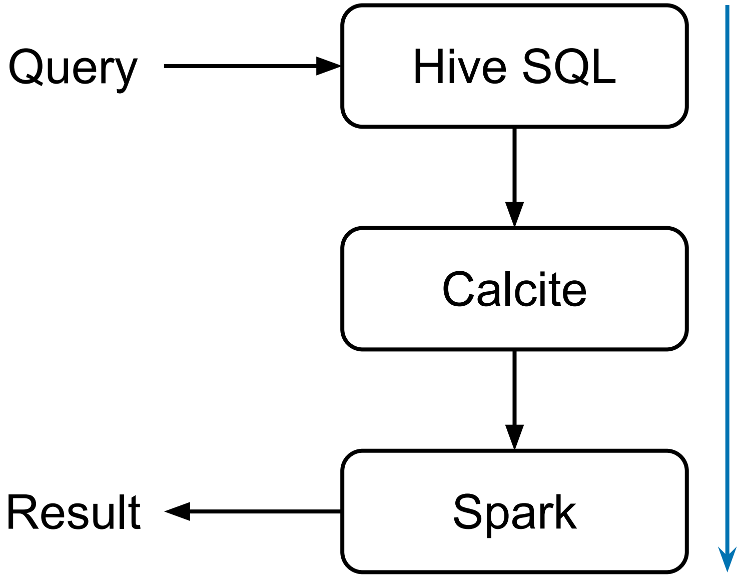
PolyDMS – Our Vision to Address the Composability Gap

- System architects should be able to build a new DMS by reusing individual components as building blocks
- We define abstract types of DMS components, such as query interfaces or optimizers, and design generic interfaces for each type so that existing libraries can easily turn into a DMS component
- We build a Component Orchestrator that interacts with all components, enabling the composition of DMSes
- We design a high-level System Definition Language (SDL) that acts as an interface to the orchestrator and allows users to easily compose a new DMS out of existing components

Existing DMS Composition Approaches



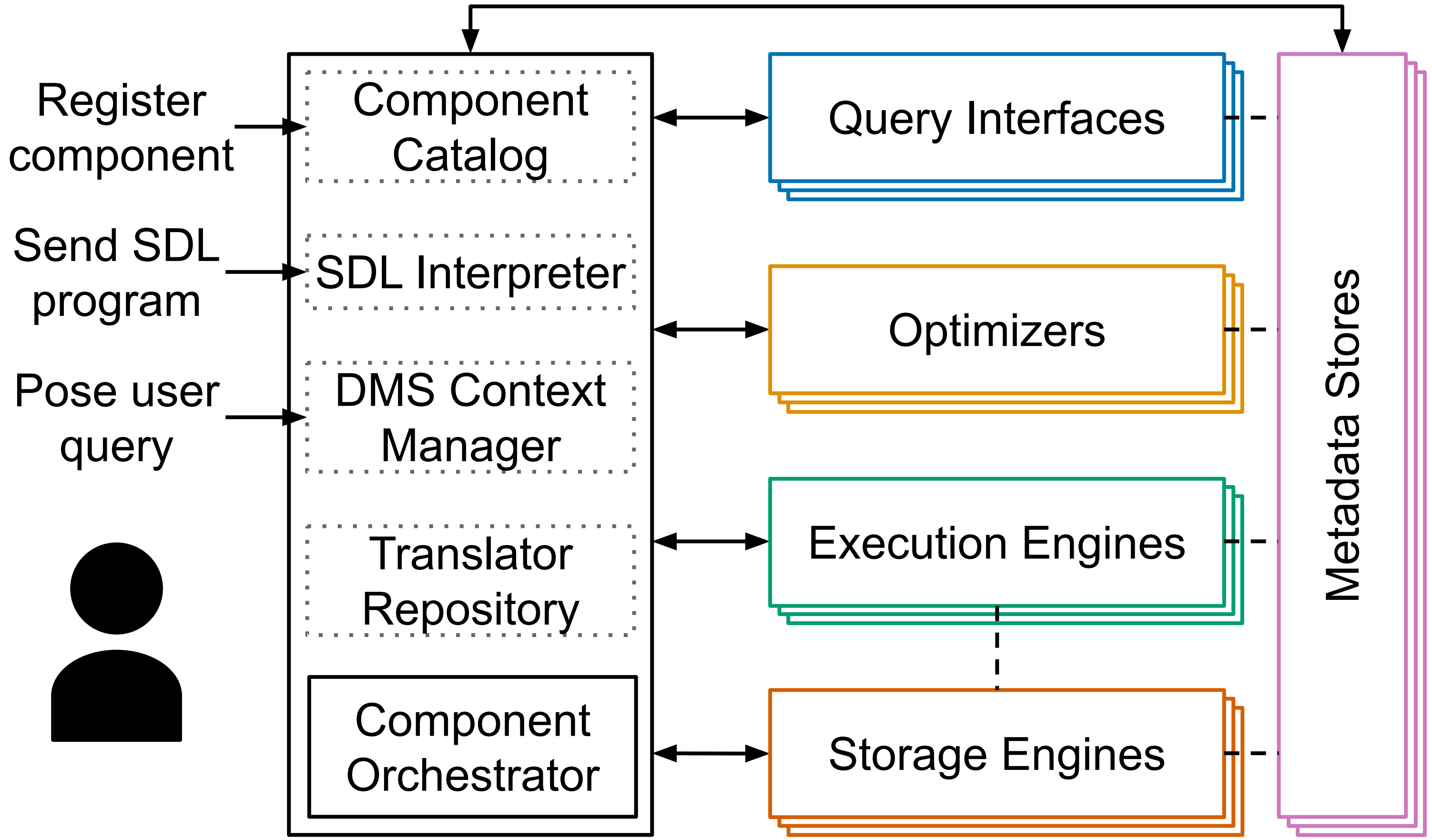
Horizontal (e.g., Wayang)



Vertical (e.g., Hive)

PolyDMS Framework

- Existing (and future) DMS components should be independently reusable
- We propose to couple components through a micro-service architecture with well-defined APIs


Register component
Send SDL program
Pose user query
Component Catalog
SDL Interpreter
DMS Context Manager
Translator Repository
Component Orchestrator
Query Interfaces
Optimizers
Execution Engines
Storage Engines
Metadata Stores

Component Types

- Metadata Stores
- Query Interfaces
- Optimizers
- Execution Engines
- Storage Engines

Component Orchestration

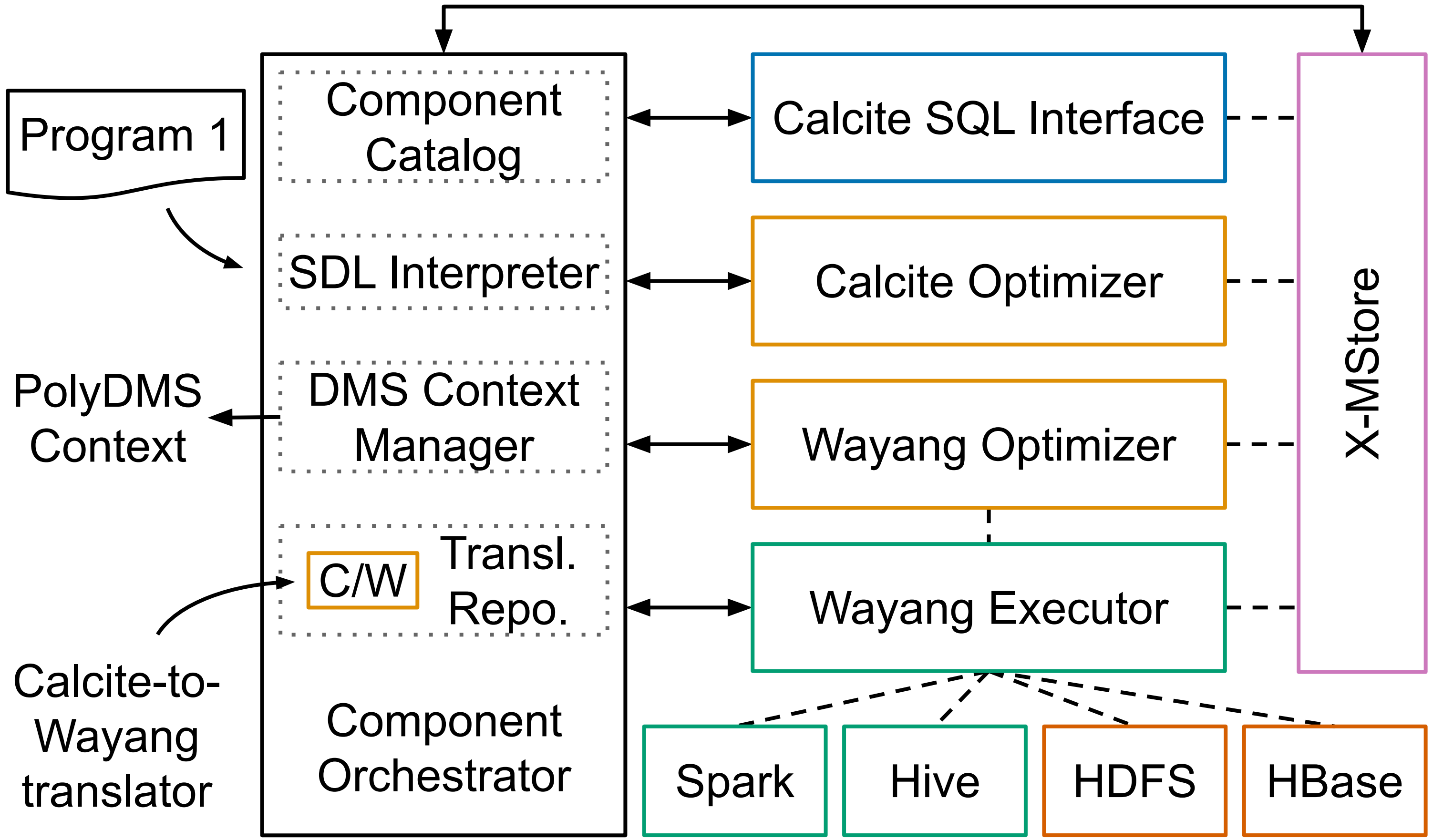
- Component Catalog
- System Definition Language (SDL) Interpreter
- DMS Context Manager
- Translator Repository

Application Scenarios

- Diverse user requirements
- Optimization synergies

Proof-of-Concept (PoC)

- Real-world industry use case that existing DMSes cannot handle efficiently in cooperation with Huawei
- Solution architects often face fixed legacy DMS infrastructures that require a cross-platform system for utilizing multiple heterogeneous DMSes in one workload
- Existing cross-platform systems do not support SQL interfaces or important logical optimizations like join ordering

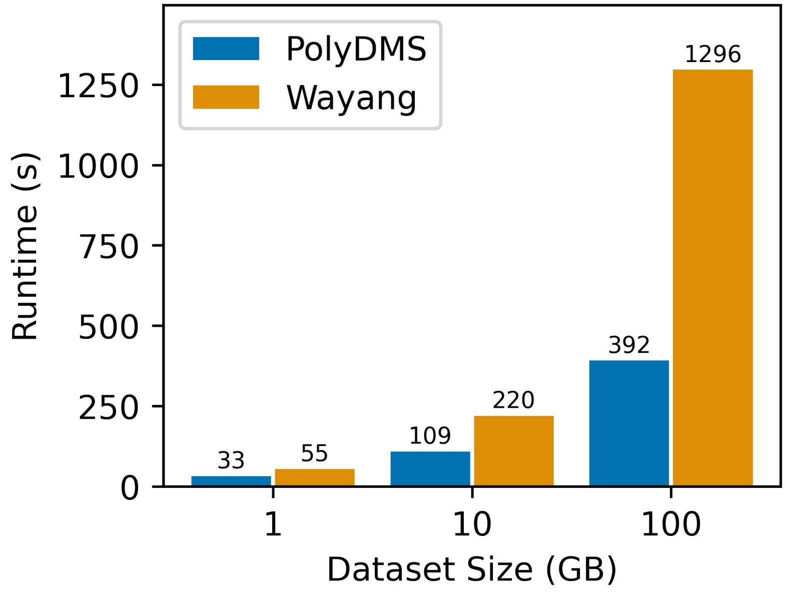

Program 1
Component Catalog
SDL Interpreter
DMS Context Manager
C/W
Transl. Repo.
Component Orchestrator
PolyDMS Context
Calcite-to-Wayang translator
Calcite SQL Interface
Calcite Optimizer
Wayang Optimizer
Wayang Executor
X-MStore
Spark
Hive
HDFS
HBase

PoC SDL Example

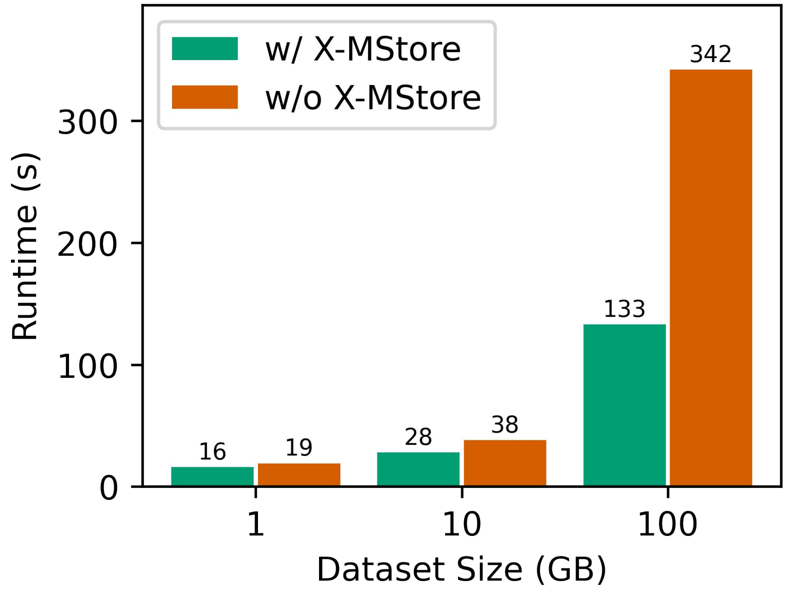
```
def cross_platform_dms(input):
    ctx = PolyDMSContext()
    ctx.md = XMStore()
    ctx.qi = CalciteSqlInterface(input, ctx.md)
    ctx.log_opt = CalciteOpt(ctx.qi, ctx.md)
    ctx.cp_opt = WayangOpt(translate(ctx.log_opt), ctx.md)
    ctx.exec = WayangExec(ctx.cp_opt, ctx.md)

    return ctx.initialize()
```

Preliminary Evaluation



Optimization synergies



Metadata impact

Outlook

- PolyDMS is our vision to break today's monolithic DMS architectures
- PolyDMS composes stand-alone components into custom DMSes
- Next: formalize our ideas for SDL and extend the orchestrator prototype to construct DMS instances based on SDL statements

Haralampos Gavriilidis
gavriilidis@tu-berlin.de
www.user.tu-berlin.de/harry_g

Lennart Behme
lennart.behme@tu-berlin.de
www.lbeh.me