Unified Data Analytics

State-of-the-art and Open Problems

Zoi Kaoudi
Jorge Quiané

Technische Universität Berlin (TU Berlin)
Current Data Analytics

SPJ Queries

```
SELECT name
FROM employee e
INNER JOIN Person p
ON p.firstname = e.name
```

Deep Learning

— Queries Beyond a Single Platform —

Q1: unstructured + structured data processing
Q2: Join Postgres data with Spark data
Q3: Run machine learning tasks over graph data
FACT 1

— One Size Does Not Fit All —
Big Data Landscape 2014
Big Data Landscape 2020
FACT 2
— Zoo of Systems —
What to Do?

Diverse Data Analytics

One Cannot Fit All

Zoo of Systems
Hybrid Systems

Not Maintainable!

Examples: HadoopDB, SystemML, FlumeJava …
Taming Them to Live Together

use of multiple data storage and/or processing platforms for data analytics

Unified Data Analytics

Unified Data

Unified Analytics
Is Unified Data Analytics New?

1. Control
   - Centralized Control

2. Degree of Integration
   - Strongly Integrated (Commonly referred to as DDBSs)

3. Degree of Heterogeneity
   - Homogeneous
   - Heterogeneous
Unified Data Analytics

Motivation

Use Cases

Summary

Federated Learning

Challenges

SOTA

Use Cases
Opportunistic to reduce the total cost of the input query.

Mandatory because the platform where the data is stored cannot run it entirely.

**Unified Data Analytics**

1. **Single-Platform** processing a query on a single data processing platform.
2. **Opportunistic** processing one single query on several data processing platforms...
3. **Mandatory** processing a query on a single data processing platform.
4. **Polystore** processing a query on several data processing platforms because the data is stored across several store engines.
Taxonomy

Unified Data Analytics

1 Single-Platform

Multi-Platform

2 Opportunistic

3 Mandatory

4 Polystore
Unified Data Analytics — Single-Platform

using *any single* data processing platform to process a query
Unified Data Analytics — Single-Platform

using **any single** data processing platform to process a query

### SGD

<table>
<thead>
<tr>
<th>Dataset size</th>
<th>Java</th>
<th>Spark</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 GB</td>
<td>290</td>
<td>414</td>
</tr>
<tr>
<td>15 GB</td>
<td>587</td>
<td>458</td>
</tr>
</tbody>
</table>

Performance Evolution

16
System-Store Quadrant

<table>
<thead>
<tr>
<th></th>
<th>Monoprocessor</th>
<th>Polyprocessor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monostore</td>
<td>Single-Platform</td>
<td></td>
</tr>
<tr>
<td>Polystore</td>
<td>Single-Platform</td>
<td></td>
</tr>
</tbody>
</table>
Taxonomy

Unified Data Analytics

1 Single-Platform

Multi-Platform

2 Opportunistic

3 Mandatory

4 Polystore
Unified Data Analytics — Opportunistic

using several data processing platforms to reduce the cost of a query

over a single data point

over a large dataset

data access

update \( w \)

sample

Parse Input

Sample

Compute

Loop

Update

SGD

APP

Java

Spark

Parse Input

Sample

Compute

Loop

Update

Parse Input

Sample

Compute

Loop

Update
Unified Data Analytics — Opportunistic

using several data processing platforms to reduce the cost of a query

SGD

Dataset size

Performance

Runtime (s)

Java  Spark  Unified

8 GB  290  22  22

15 GB  587  458  24
## System-Store Quadrant

<table>
<thead>
<tr>
<th></th>
<th>Monoplatinum</th>
<th>Polyplatform</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Monostore</strong></td>
<td><em>Single-Platform</em></td>
<td><em>Opportunistic</em></td>
</tr>
<tr>
<td><strong>Polystore</strong></td>
<td><em>Single-Platform</em></td>
<td></td>
</tr>
</tbody>
</table>
Taxonomy

Unified Data Analytics

1. Single-Platform
2. Opportunistic
3. Mandatory
4. Polystore
Unified Data Analytics — Mandatory

using *several* data processing platforms to be able to process a query
# System-Store Quadrant

<table>
<thead>
<tr>
<th></th>
<th>Monoplatfrom</th>
<th>Polyplatform</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Monostore</strong></td>
<td><strong>Single-Platform</strong></td>
<td><strong>Opportunistic</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Mandatory</strong></td>
</tr>
<tr>
<td><strong>Polystore</strong></td>
<td><strong>Single-Platform</strong></td>
<td></td>
</tr>
</tbody>
</table>
Unified Data Analytics — Polystore

using *several* data processing platforms because data is spread

Join

Customer

Orders

Map

Join

Map

Join

Map

Join jr1 with line item

join customer with orders

jr1

data access

data access
<table>
<thead>
<tr>
<th>Monostore</th>
<th>Polystore</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monoprocessor</td>
<td>Polystore</td>
</tr>
<tr>
<td>Single-Platform</td>
<td>Opportunistic</td>
</tr>
<tr>
<td>Mandatory</td>
<td>Polystore</td>
</tr>
</tbody>
</table>

**System-Store Quadrant**
Questions?
Challenges

1. Decoupling Applications
2. Data Movement
3. Automatic Unified Data Analytics
4. Extensibility
Unified Data Analytics

Motivation

Use Cases

Challenges

SOTA

Federated Learning

Summary
Challenges

1. Decoupling Applications
2. Data Movement
3. Automatic Unified Data Analytics
4. Extensibility
Decoupling Applications

Applications/ Frontends
- Hive
- Crunch
- MLlib
- Mahout
- Pig

Processing Platforms
- Hadoop MR
- DBMS
- Storm
- Spark
- Flink

Storage Engines
- HDFS
- S3
- Local FS
Decoupling Applications

Unified Data Analytics System

Applications/Frontends
- Hive
- Crunch
- MLlib
- Mahout
- Pig

Processing Platforms
- Hadoop
- DBMS
- Storm
- MR
- Spark
- Flink

Storage Engines
- HDFS
- S3
- Local FS
Decoupling Applications

Applications/Frontends
- Hive
- Crunch
- MLlib
- Mahout
- Pig

Processing Platforms
- Hadoop MR
- DBMS
- Storm
- Spark
- Flink

Storage Engines
- HDFS
- S3
- Local FS

Musketeer
- Apache Beam
- Apache Wayang
- BigDawg
- Ires
- DBMS+
Decoupling Applications

Applications/Frontends
- Hive
- Crunch
- MLlib
- Mahout
- Pig

Processing Platforms
- Hadoop
- MR
- DBMS
- Spark
- Flink

Storage Engines
- HDFS
- S3
- Local FS

Musketeer
Apache Beam
BigDawg
Apache Wayang
Ires
DBMS+
Musketeer

DAG-based intermediate representation

Front-ends

Hive
Pig
DryadLINQ
GAS DSL
GreenMarl
SQL DSL
SparkSQL
GraphX

Intermediate representation

Back-ends

Hadoop
Metis
Dryad
Spark
Naiad
CIEL
Giraph
PowerGraph
Chi
Stream

SQL / vertex-centric graph abstractions

code templates

Musketeer: all for one, one for all in data processing systems. EuroSys’15.
## Decoupling Features

<table>
<thead>
<tr>
<th>Processing Model</th>
<th>Granularity</th>
<th>Mappings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Musketeer</td>
<td>Dataflow DAG + Loop</td>
<td>Operator</td>
</tr>
</tbody>
</table>
Apache Wayang (incubating) & Ires

2 layers of operators

Wayang operators (Wayang)
Abstract operators (Ires)

Execution operators (Wayang)
Materialized operators (Ires)

Wayang: fine-granular operators

Ires: Task-granular operators

Road to Freedom in Big Data Analytics. EDBT 2016
Mix ’n’ Match Multi-Engine Analytics. IEEE BigData 2017
Apache Wayang (incubating) & Ires

**Graph-based** mappings

Wayang operators

<table>
<thead>
<tr>
<th>Execution operators</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReduceBy</td>
</tr>
<tr>
<td>SparkReduceBy</td>
</tr>
<tr>
<td>GroupBy</td>
</tr>
<tr>
<td>JavaGroupBy</td>
</tr>
<tr>
<td>Map</td>
</tr>
<tr>
<td>JavaMap</td>
</tr>
</tbody>
</table>

**Metadata-based** mappings

Abstract operators

<table>
<thead>
<tr>
<th>TF_IDF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
</tr>
<tr>
<td>number : 1</td>
</tr>
<tr>
<td>Constraints</td>
</tr>
<tr>
<td>EngineSpecification</td>
</tr>
<tr>
<td>Engine</td>
</tr>
<tr>
<td>Distributed</td>
</tr>
<tr>
<td>MapReduce</td>
</tr>
<tr>
<td>masterLocation : 127.0.0.1</td>
</tr>
<tr>
<td>Algorithm</td>
</tr>
<tr>
<td>name : TF_IDF</td>
</tr>
<tr>
<td>Materialized operators</td>
</tr>
<tr>
<td>TF_IDF_mahout</td>
</tr>
<tr>
<td>Engine</td>
</tr>
<tr>
<td>FS : HDFS</td>
</tr>
<tr>
<td>Type : SequenceFile</td>
</tr>
<tr>
<td>Output</td>
</tr>
<tr>
<td>number : 1</td>
</tr>
<tr>
<td>Execution</td>
</tr>
<tr>
<td>FS : HDFS</td>
</tr>
<tr>
<td>Type : SequenceFile</td>
</tr>
</tbody>
</table>

To discover the actual implementations that comply with the planning and optimization phase, described subsequently.
## Decoupling Features

<table>
<thead>
<tr>
<th>Processing Model</th>
<th>Granularity</th>
<th>Mappings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Musketeer</td>
<td>Dataflow DAG + Loop</td>
<td>Operator</td>
</tr>
<tr>
<td>Apache Wayang</td>
<td>Dataflow DAG + Loop</td>
<td>Operator</td>
</tr>
<tr>
<td>Ires</td>
<td>Workflow graph</td>
<td>Task</td>
</tr>
</tbody>
</table>
Apache Beam

Translators using 1:n mappings

Beam Pipeline

Flink Runner

Pipeline Translators

Flink Job

- Flatten
- Union
- Combine
- CombineByKey
- Aggregate

https://beam.apache.org/
## Decoupling Features

<table>
<thead>
<tr>
<th>Processing Model</th>
<th>Granularity</th>
<th>Mappings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Musketeer</td>
<td>Dataflow DAG + Loop</td>
<td>Operator</td>
</tr>
<tr>
<td>Apache Wayang</td>
<td>Dataflow DAG + Loop</td>
<td>Operator</td>
</tr>
<tr>
<td>Ires</td>
<td>Workflow graph</td>
<td>Task</td>
</tr>
<tr>
<td>Apache Beam</td>
<td>Pipeline (DAG)</td>
<td>Operator</td>
</tr>
<tr>
<td>Processing Model</td>
<td>Granularity</td>
<td>Mappings</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-------------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>Musketeer</td>
<td>Dataflow DAG + Loop</td>
<td>Operator</td>
</tr>
<tr>
<td>Apache Wayang</td>
<td>Dataflow DAG + Loop</td>
<td>Operator</td>
</tr>
<tr>
<td>Ires</td>
<td>Workflow graph</td>
<td>Task</td>
</tr>
<tr>
<td>Apache Beam</td>
<td>Pipeline (DAG)</td>
<td>Operator</td>
</tr>
<tr>
<td>Myria</td>
<td>Extended relational</td>
<td>Operator</td>
</tr>
<tr>
<td>DBMS+/Cyclops</td>
<td>Windowed aggregation</td>
<td>Operator</td>
</tr>
<tr>
<td>BigDawg</td>
<td>Query Plan Tree</td>
<td>Operator</td>
</tr>
</tbody>
</table>
Challenges

1. Decoupling Applications
2. Data Movement
3. Automatic Unified Data Analytics
4. Extensibility
Musketeer: all for one, one for all in data processing systems. EuroSys’15.
Data Movement Features

<table>
<thead>
<tr>
<th>Musketeer</th>
<th>Connection</th>
<th>Transfer Means</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Direct</td>
<td>HDFS Files</td>
</tr>
</tbody>
</table>
Connecting two platforms
Connecting two platforms

Inefficient
PipeGen by Myria

**Binary** data transfer via generated **data pipes**

**Same Import/Export Capabilities**

Optimizations:
1. Binary encoding
2. Compression
3. Columnar form

Program Analysis

PipeGen: Data Pipe Generator for Hybrid Analytics. SoCC 2016.
## Data Movement Features

<table>
<thead>
<tr>
<th>Connection</th>
<th>Transfer Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>Musketeer</td>
<td>Direct</td>
</tr>
<tr>
<td></td>
<td>HDFS Files</td>
</tr>
<tr>
<td>Myria</td>
<td>Direct</td>
</tr>
<tr>
<td></td>
<td>Generated data pipes</td>
</tr>
</tbody>
</table>
Channels Graph in Apache Wayang

Group Steiner Tree Problem

Channel conversion graph

reusable channel
non-reusable channel

# Data Movement Features

<table>
<thead>
<tr>
<th>Connection</th>
<th>Transfer Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>Musketeer</td>
<td>Direct</td>
</tr>
<tr>
<td></td>
<td>HDFS Files</td>
</tr>
<tr>
<td>Myria</td>
<td>Direct</td>
</tr>
<tr>
<td></td>
<td>Generated data pipes</td>
</tr>
<tr>
<td>Apache Wayang</td>
<td>Graph</td>
</tr>
<tr>
<td></td>
<td>Conversion operators</td>
</tr>
</tbody>
</table>
## Data Movement Features

<table>
<thead>
<tr>
<th>Connection</th>
<th>Transfer Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>Musketeer</td>
<td>Direct</td>
</tr>
<tr>
<td></td>
<td>HDFS Files</td>
</tr>
<tr>
<td>Myria</td>
<td>Direct</td>
</tr>
<tr>
<td></td>
<td>Generated data pipes</td>
</tr>
<tr>
<td>Apache Wayang</td>
<td>Graph</td>
</tr>
<tr>
<td></td>
<td>Conversion operators</td>
</tr>
<tr>
<td>Ires</td>
<td>Direct</td>
</tr>
<tr>
<td></td>
<td>Move operators</td>
</tr>
<tr>
<td>DBMS+</td>
<td>Direct</td>
</tr>
<tr>
<td></td>
<td>HDFS Files</td>
</tr>
</tbody>
</table>
## Data Movement Features

<table>
<thead>
<tr>
<th>Connection</th>
<th>Transfer Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>Musketeer</td>
<td>Direct</td>
</tr>
<tr>
<td></td>
<td>HDFS Files</td>
</tr>
<tr>
<td>Myria</td>
<td>Direct</td>
</tr>
<tr>
<td></td>
<td>Generated data pipes</td>
</tr>
<tr>
<td>Apache Wayang</td>
<td>Graph</td>
</tr>
<tr>
<td></td>
<td>Conversion operators</td>
</tr>
<tr>
<td>Ires</td>
<td>Direct</td>
</tr>
<tr>
<td></td>
<td>Move operators</td>
</tr>
<tr>
<td>DBMS+</td>
<td>Direct</td>
</tr>
<tr>
<td></td>
<td>HDFS Files</td>
</tr>
</tbody>
</table>

*Inefficient*
## Data Movement Features

<table>
<thead>
<tr>
<th></th>
<th>Connection</th>
<th>Transfer Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>Musketeer</td>
<td>Direct</td>
<td>HDFS Files</td>
</tr>
<tr>
<td>Myria</td>
<td>Direct</td>
<td>Generated data pipes</td>
</tr>
<tr>
<td>Apache Wayang</td>
<td>Graph</td>
<td>Conversion operators</td>
</tr>
<tr>
<td>Ires</td>
<td>Direct</td>
<td>Move operators</td>
</tr>
<tr>
<td>DBMS+</td>
<td>Direct</td>
<td>HDFS Files</td>
</tr>
</tbody>
</table>

- Not scalable
- Inefficient
Questions?
Challenges

1. Decoupling Applications
2. Data Movement
3. Automatic Unified Data Analytics
4. Extensibility
Decoupling Applications

Applications/ Frontends
- Hive
- Crunch
- MLlib
- Mahout
- Pig

Unified Data Analytics System

Processing Platforms
- Hadoop
  - MR
- DBMS
- Storm
- Spark
- Flink

Storage Engines
- HDFS
- S3
- Local FS
Which Platforms to Use?
Unified Data Analytics Query Optimization

Rule-based

Cost-based
Unified Data Analytics Query Optimization

Rule-based  
Cost-based  
Learning-based
Musketeer

Our intermediate representation is a dynamic directed acyclic graph (DAG) from which we generate jobs for back-end execution engines. This information may not be available at workflow implementation time, which motivates our approach of decoupling execution engines from front-ends.

We believe that a decoupled data processing architecture gives users additional flexibility. In this approach, the front-end workflow specification is translated into an intermediate representation from which we generate jobs for back-end execution engines. This is common in industry: up to 80% of jobs running in production clusters come from front-end frameworks. This is in contrast to the tightly coupled "all for one, one for all" approach that we advocate.

Musketeer is our proof-of-concept implementation of the decoupled "all for one, one for all" approach that we advocate. It translates a workflow defined in a front-end framework such as Pig [35], Hive [41], Shark [8], Spark [21], Naiad [5], MetaSched [32], DryadLINQ [38], and Dryad [2] to a common form; we do this by generating jobs for the back-end execution engine Hadoop [4]. Our approach is extensible: new front-end frameworks can be added by providing translation logic, and new back-end execution engines can be chosen execution engines and back-ends automatically.

Limitations. Our approach would simply introduce a heuristic to solve it efficiently for large DAGs. This information may not be available at workflow implementation time, which motivates our approach of decoupling execution engines from front-ends.

Finally, we break the execution of a data processing workflow into three layers: front-end, intermediate representation, and back-end. The front-end is responsible for describing their realization in our Musketeer prototype. The front-end workflow specifications are generated from this representation and executed on one back-end execution engine. This information may not be available at workflow implementation time, which motivates our approach of decoupling execution engines from front-ends.

How to map the IR to back-ends automatically?
Musketeer: all for one, one for all in data processing systems. EuroSys’15.
## Optimization Features

<table>
<thead>
<tr>
<th>Method</th>
<th>Enumeration</th>
<th>Cost Model</th>
<th>Data Movement Cost</th>
<th>Bypass Optimizer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Musketeer</td>
<td>Cost-based</td>
<td>Exhaustive/DP</td>
<td>Analytical</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>No</td>
</tr>
</tbody>
</table>
Apache Wayang (incubating)

Wayang plan
(platform-agnostic)

Plan Inflation

Wayang’s optimizer

Apache Wayang (incubating)

Plan Inflation

Wayang plan

Mappings

Inflated plan

Apache Wayang (incubating)

Wayang plan (platform-agnostic)

Plan Inflation

Operator Costs

Movement Costs

Plan Enumeration

Wayang’s optimizer

Inflated plan + operator costs + data movement costs

Learning-based coefficients of cost model

cheapest execution plan

## Optimization Features

<table>
<thead>
<tr>
<th>Method</th>
<th>Enumeration</th>
<th>Cost Model</th>
<th>Data Movement Cost</th>
<th>Bypass Optimizer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Musketeer</td>
<td>Cost-based</td>
<td>Analytical</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Apache Wayang</td>
<td>Cost-based</td>
<td>Enumeration algebra</td>
<td>Yes</td>
<td>Operator level</td>
</tr>
<tr>
<td></td>
<td>Exhaustive/DP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Analytical</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Analytical with learned coefficients</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Optimization Features

<table>
<thead>
<tr>
<th>Method</th>
<th>Enumeration</th>
<th>Cost Model</th>
<th>Data Movement Cost</th>
<th>Bypass Optimizer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Musketeer</td>
<td>Cost-based</td>
<td>Exhaustive/DP</td>
<td>Analytical</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Apache Wayang</td>
<td>Cost-based</td>
<td>Enumeration algebra</td>
<td>Analytical with learned coefficients</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ires</td>
<td>Cost-based</td>
<td>DP</td>
<td>Learned per operator</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cyclops</td>
<td>Cost-based</td>
<td>?</td>
<td>Learned per system</td>
<td>No</td>
</tr>
</tbody>
</table>

The table above summarizes the optimization features of various systems. Musketeer uses a cost-based method, exhaustive/DP enumeration, and an analytical cost model with no bypass optimizer. Apache Wayang employs a cost-based method, enumeration algebra, and an analytical cost model with learned coefficients, offering a bypass optimizer at the operator level. Ires uses a cost-based method, DP enumeration, and a learned per operator cost model, providing a bypass optimizer at the operator level. Cyclops also uses a cost-based method, with a learned per system cost model, and no bypass optimizer, offering a task level approach.
Unified Data Analytics Optimization

- Cost-based
- Rule-based
- Learning-based
Problems with cost-based optimizer

- Low control
- Tedious definition of cost formulas
- Assume linear functions
- Hard to fine-tune

Cost_{sparkjoin} = a_1 x_{cpu} + a_2 x_{memory} + a_n x_{network} + \ldots

ML-based Cross-Platform Query Optimization. ICDE 2020: 1489-1500
Effect of Cost Model Tuning

More than an order of magnitude better performance

Using real cardinalities

Well-tuned cost model
Simply-tuned cost model

Runtime (sec)
From Cost Model to ML Model

Plan Enumeration

Search space

Physical operators

Logical plan

Execution plan

Cost model

Statistics
Plan Enumeration

From Cost Model to ML Model
<table>
<thead>
<tr>
<th>Method</th>
<th>Enumeration</th>
<th>Cost Model</th>
<th>Data Movement Cost</th>
<th>Bypass Optimizer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Musketeer</td>
<td>Cost-based</td>
<td>Exhaustive/DP</td>
<td>Analytical</td>
<td>No</td>
</tr>
<tr>
<td>Apache Wayang</td>
<td>Cost-based</td>
<td>Enumeration algebra</td>
<td>ML-based</td>
<td>Yes</td>
</tr>
<tr>
<td>Ires</td>
<td>Cost-based</td>
<td>DP</td>
<td>ML-based</td>
<td>Yes</td>
</tr>
<tr>
<td>Cyclops</td>
<td>Cost-based</td>
<td>?</td>
<td>ML-based</td>
<td>No</td>
</tr>
<tr>
<td>Apache Wayang</td>
<td>Learning-based</td>
<td>Vector-based enumeration algebra</td>
<td>ML model</td>
<td>Yes</td>
</tr>
<tr>
<td>Myria</td>
<td>Rule-based</td>
<td>Bottom up</td>
<td>NA</td>
<td>NA</td>
</tr>
</tbody>
</table>
## Optimization Features

<table>
<thead>
<tr>
<th>Method</th>
<th>Enumeration</th>
<th>Cost Model</th>
<th>Data Movement Cost</th>
<th>Bypass Optimizer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Musketeer</td>
<td>Cost-based</td>
<td>Exhaustive/DP</td>
<td>Analytical</td>
<td>No</td>
</tr>
<tr>
<td>Apache Wayang</td>
<td>Cost-based</td>
<td>Enumeration algebra</td>
<td>ML-based</td>
<td>Yes</td>
</tr>
<tr>
<td>Ires</td>
<td>Cost-based</td>
<td>DP</td>
<td>ML-based</td>
<td>Yes</td>
</tr>
<tr>
<td>Cyclops</td>
<td>Cost-based</td>
<td>?</td>
<td>ML-based</td>
<td>No</td>
</tr>
<tr>
<td>Apache Wayang</td>
<td>Learning-based</td>
<td>Vector-based enumeration algebra</td>
<td>ML model</td>
<td>Yes</td>
</tr>
<tr>
<td>Myria</td>
<td>Rule-based</td>
<td>NA</td>
<td>No</td>
<td></td>
</tr>
</tbody>
</table>

*Not scalable*
## Optimization Features

<table>
<thead>
<tr>
<th>Method</th>
<th>Enumeration</th>
<th>Cost Model</th>
<th>Data Movement Cost</th>
<th>Bypass Optimizer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Musketeer</td>
<td>Cost-based</td>
<td>Exhaustive/DP</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Apache Wayang</td>
<td>Cost-based</td>
<td>Enumeration</td>
<td>Yes</td>
<td>Operator level</td>
</tr>
<tr>
<td>Ires</td>
<td>Cost-based</td>
<td></td>
<td>Yes</td>
<td>Operator level</td>
</tr>
<tr>
<td>Cyclops</td>
<td>Cost-based</td>
<td>?</td>
<td>No</td>
<td>Task level</td>
</tr>
<tr>
<td>Apache Wayang</td>
<td>Learning-based</td>
<td>Vector-based</td>
<td>Yes</td>
<td>Operator level</td>
</tr>
<tr>
<td>Myria</td>
<td>Rule-based</td>
<td>NA</td>
<td>No</td>
<td></td>
</tr>
</tbody>
</table>

**Hard to fine-tune**

**Not scalable**
## Optimization Features

<table>
<thead>
<tr>
<th>Method</th>
<th>Enumeration</th>
<th>Cost Model</th>
<th>Data Movement Cost</th>
<th>Bypass Optimizer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Musketeer</td>
<td>Cost-based</td>
<td>Exhaustive/DP</td>
<td>Analytical</td>
<td>No</td>
</tr>
<tr>
<td>Apache Wayang</td>
<td>Cost-based</td>
<td>Enumeration</td>
<td>ML-based</td>
<td>Yes</td>
</tr>
<tr>
<td>Ires</td>
<td>Cost-based</td>
<td>?</td>
<td>ML-based</td>
<td>Yes</td>
</tr>
<tr>
<td>Cyclops</td>
<td>Cost-based</td>
<td>?</td>
<td>ML-based</td>
<td>No</td>
</tr>
<tr>
<td>Apache Wayang</td>
<td>Learning-based</td>
<td>?</td>
<td>ML-based</td>
<td>Yes</td>
</tr>
<tr>
<td>Myria</td>
<td>Rule-based</td>
<td></td>
<td></td>
<td>NA</td>
</tr>
</tbody>
</table>

- **Hard to fine-tune**
- **Training data**
- **Not scalable**
Challenges

1. Decoupling Applications

2. Data Movement

3. Automatic Unified Data Analytics

4. Extensibility
Musketeer

**Code templates** describing a platform

Front-ends
- Hive
- Pig
- DryadLINQ
- GAS DSL
- GreenMarl
- SQL DSL
- SparkSQL
- GraphX
- Lindi
- GraphLINQ

Intermediate representation

Back-ends
- Hadoop
- Metis
- Dryad
- Spark
- Naiad
- CIEL
- Giraph
- PowerGraph
- GraphChi
- X-Stream

+ cost model

Musketeer: all for one, one for all in data processing systems. EuroSys’15.
Apache Wayang

Execution operators describing a platform

Wayang Operators $\rightarrow$ Wayang $\rightarrow$ mappings $\rightarrow$ Execution Operators

**RHEEMix in the data jungle: a cost-based optimizer for cross-platform systems.** VLDB J. 29(6): 1287-1310 (2020)
Apache Wayang

Wayang Operators

Execution Operators

Wayang mappings

## Extensibility Features

<table>
<thead>
<tr>
<th>Platform</th>
<th>Description</th>
<th>IR Extensibility</th>
<th>Difficulty</th>
</tr>
</thead>
<tbody>
<tr>
<td>Musketeer</td>
<td>Code Templates</td>
<td>Internal code changes</td>
<td>High</td>
</tr>
<tr>
<td>Apache Wayang</td>
<td>Execution Operators</td>
<td>Wayang operators</td>
<td>Medium</td>
</tr>
<tr>
<td>Myria</td>
<td>AST</td>
<td>—</td>
<td>Medium</td>
</tr>
</tbody>
</table>
Unified Data Analytics

- Motivation
- Use Cases
- Challenges
- SOTA
- Summary
- Federated Learning
Current Data Analytics

SPJ Queries

```
SELECT name
FROM employee e
INNER JOIN Person p
ON p.firstname = e.name
```

Relational Data

Graph Data
Current Data Analytics

— Machine Learning —

— Federated Learning —
What is Federated Learning?

Train ML models across many devices without centralized data collection
Federated Learning as Unified Data Analytics

Automatic Unified Data Analytics

Local Model Updates

Local training

Global Model
Challenges in Federated Learning

1. How to aggregate partial results
2. Model compatibility
3. Client selection
4. Privacy guarantees

Automatic Unified Data Analytics
Some Federated Learning Systems

Tensorflow
Federated

Flower

Syft

FATE
Some Federated Learning Systems

Tensorflow
Federated

Federated Aggregation

TFF models have to be serializable as TF graph

Only local simulations
Some Federated Learning Systems

Syft aggregator

Syft

PyTorch
Some Federated Learning Systems

ML framework-agnostic

Flower engine

Heterogeneous clients
Unified Data Analytics

- Motivation
- Use Cases
- Challenges
- SOTA
- Federated Learning
- Summary
<table>
<thead>
<tr>
<th>System</th>
<th>Processing Model</th>
<th>Optimizer</th>
<th>Data Movement Cost</th>
<th>Extensibility Effort</th>
</tr>
</thead>
<tbody>
<tr>
<td>Musketeer</td>
<td>Dataflow DAG + Loop</td>
<td>Yes</td>
<td>No</td>
<td>High</td>
</tr>
<tr>
<td>Apache Wayang</td>
<td>Dataflow DAG + Loop</td>
<td>Yes</td>
<td>Yes</td>
<td>Medium</td>
</tr>
<tr>
<td>BigDawg</td>
<td>Query Plan</td>
<td>NA</td>
<td>NA</td>
<td>High</td>
</tr>
<tr>
<td>Myria</td>
<td>Extended</td>
<td>NA</td>
<td>NA</td>
<td>High</td>
</tr>
<tr>
<td>Cyclops</td>
<td>Windowed aggregation</td>
<td>Yes</td>
<td>Yes</td>
<td>High</td>
</tr>
<tr>
<td>Apache Beam</td>
<td>Dataflow DAG</td>
<td>No</td>
<td>NA</td>
<td>High</td>
</tr>
<tr>
<td>ML Systems</td>
<td>ML dataflow</td>
<td>No</td>
<td>NA</td>
<td>High/Medium</td>
</tr>
</tbody>
</table>

**Recap**

- **Fixed Processing Model**

- **Time-Consuming & Hard Cost Tuning**
### Recap

<table>
<thead>
<tr>
<th>Processing Model</th>
<th>Optimizer</th>
<th>Data Movement Cost</th>
<th>Extensibility Effort</th>
</tr>
</thead>
<tbody>
<tr>
<td>Musketeer</td>
<td>Dataflow DAG + Loop</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Apache Wayang</td>
<td>Dataflow DAG + Loop</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>BigDawg</td>
<td>Query Plan + Loop</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Myria</td>
<td>Extended</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Cyclops</td>
<td>Windowed aggregation</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Apache Beam</td>
<td>Dataflow DAG</td>
<td>No</td>
<td>NA</td>
</tr>
<tr>
<td>ML Systems</td>
<td>ML flows</td>
<td>No</td>
<td>NA</td>
</tr>
</tbody>
</table>
Open Problems

Flexible Processing Model
How to support arbitrary processing models?

Fully Automatic Cost Tuning
How to match real workloads without knowing them?

Precise Cardinality Estimation
Weak control over the underlying platforms

Platform Integration
How to integrate a platform fully automatically?

Fast Data Transfer
What is the right intermediate data representation?
Take Away

real need for **data processing independence**

<table>
<thead>
<tr>
<th>Monostore</th>
<th>Polystore</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Monoplatfor</strong>n</td>
<td><strong>Polyplatform</strong></td>
</tr>
<tr>
<td>Single-Platform</td>
<td>Opportunistic</td>
</tr>
<tr>
<td></td>
<td>Mandatory</td>
</tr>
<tr>
<td>Single-Platform</td>
<td>Polystore</td>
</tr>
</tbody>
</table>

---

**Queries Beyond a Single Platform**

---

- **Musketre**
  - Dataflow DAG
  - Loop
  - CP Optimizer: Yes
  - Data Movement Cost: No
  - Extensibility: High

- **Rheem**
  - Dataflow DAG
  - CP Optimizer: Yes
  - Data Movement Cost: Yes
  - Extensibility: Medium

---

**No Perfect Current Solution**

---

- **Sgd**
  - Dataflow DAG
  - CP Optimizer: No
  - Data Movement Cost: NA
  - Extensibility: High