SHORT-PAPER

Apache Wayang in Action: Enabling Data Systems Integration via a Unified Data Analytics Framework

KAUSTUBH BEEDKAR, Indian Institute of Technology Delhi, New Delhi, DL, India

AURÉLIEN BERTRAND, IT University of Copenhagen, Copenhagen, Denmark

HARALAMPOS GAVRIILIDIS, Technical University of Berlin, Berlin, Germany

AUGUSTO FONSECA

ZOI KAOUDI, IT University of Copenhagen, Copenhagen, Denmark

MINGXI LIU, East China Normal University, Shanghai, China

View all

Published: 22 June 2025

SIGMOD/PODS '25: International Conference on Management of Data
June 22 - 27, 2025
Berlin, Germany

Conference Sponsors:
SIGMOD

https://doi.org/10.1145/3722212.3725081
ISBN: 9798400715648

Apache Wayang in Action: Enabling Data Systems Integration via a Unified Data Analytics Framework

Kaustubh Beedkar
Indian Institute of Technology Dehli
New Delhi, India

Aurélien Bertrand
IT University of Copenhagen
Copenhagen, Denmark

Haralampos Gavriilidis
TU Berlin
Berlin, Germany

Augusto Fonseca
LNCC-DEXL
Petrópolis, Brazil

Zoi Kaoudi
IT University of Copenhagen
Copenhagen, Denmark

Mingxi Liu
East China Normal University
Shanghai, China

Volker Markl
BIFOLD, TU Berlin, DFKI
Berlin, Germany

Juri Petersen
IT University of Copenhagen
Copenhagen, Denmark

Fabio Porto
LNCC-DEXL
Petrópolis, Brazil

Víctor Ribeiro
LNCC-DEXL
Petrópolis, Brazil

Mads Sejer Pedersen
IT University of Copenhagen
Copenhagen, Denmark

Lucas Tavares
LNCC-DEXL
Petrópolis, Brazil

Michalis Vargiamis
Scalytics
Miami, Florida, USA

Chen Xu
East China Normal University
Shanghai, China

Abstract

Apache Wayang is an open-source framework, which provides a systematic and efficient solution for unifying data analytics over disparate data sources and via integrating multiple heterogeneous data systems. It achieves that by decoupling applications from the underlying systems. In addition, it provides an optimizer so that users do not have to specify the platforms on which their pipeline should run but the optimizer can determine the best way given a cost metric. In this demonstration, we showcase how the flexible architecture of Wayang enables seamless integration with multiple heterogeneous data systems and how the query optimizer can lead to better performance.

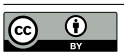
CCS Concepts

• **Information systems** → **Data management systems**; **Query optimization**; **DBMS engine architectures**.

Keywords

Unified Data Analytics, Query Optimization

ACM Reference Format:
Kaustubh Beedkar, Aurélien Bertrand, Haralampos Gavriilidis, Augusto Fonseca, Zoi Kaoudi, Mingxi Liu, Volker Markl, Juri Petersen, Fabio Porto, Víctor Ribeiro, Mads Sejer Pedersen, Lucas Tavares, Michalis Vargiamis, and Chen Xu. 2025. Apache Wayang in Action: Enabling Data Systems Integration via a Unified Data Analytics Framework. In *Companion of the 2025 International Conference on Management of Data (SIGMOD-Companion '25), June 22–27, 2025, Berlin, Germany*. ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3722212.3725081



SIGMOD-Companion '25, Berlin, Germany

ACM ISBN 979-8-4007-1564-8/2025/06
https://doi.org/10.1145/3722212.3725081

1 Introduction

Research and industry communities have developed many data systems to derive value from data efficiently. Each system possesses distinct strengths within the design space: Big data processing systems excel in batch processing large amounts of data, other systems specialize in performing graph analytics, while lately, a large number of systems and libraries aim at making machine learning and AI more efficient and accessible. Consequently, users are confronted with a myriad of specialized data systems for conducting data analytics, leading to higher operational costs due to the daunting task of selecting the most suitable one. Moreover, modern applications often demand analytics capabilities that surpass individual systems' limitations, leading to expensive ETL processes. This exacerbates the complexity of data system selection.

There is a growing need for unifying data analytics within a single framework [3, 4] to alleviate the challenges associated with data system selection and integration. Unified data analytics enables applications to operate seamlessly and efficiently across diverse systems. The necessity of data system integration for unifying data analytics arises across a spectrum of tasks, ranging from simple operations to intricate processes such as comprehensive data science pipelines involving data cleaning, preparation, feature extraction, and model training. Unified data analytics is swiftly evolving into a fundamental requirement as new applications emerge.

Apache Wayang [1, 2] stands as the first open-source middleware offering seamless data system integration and query optimization

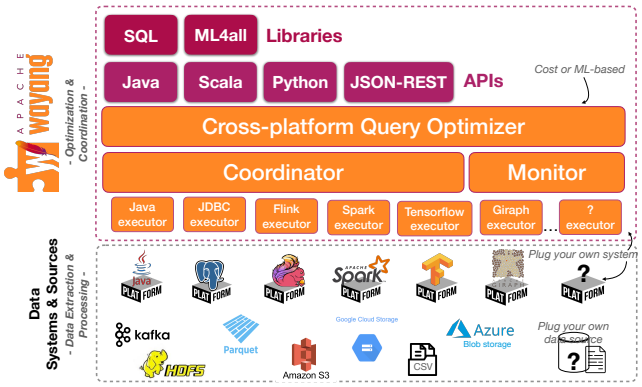


Figure 1: Wayang architecture: Data extraction and processing is done by the underlying data systems while Wayang performs system selection and coordination.

for unified data analytics. Wayang achieves this by decoupling applications from specific systems, enabling them to execute analytics seamlessly and efficiently across one or more systems. Developers can programmatically define their data analytics tasks with Java, Scala, or Python or declaratively through SQL. Wayang receives an input task and generates an execution plan that may span multiple systems. This is done via its query optimizer with the goal of minimizing the cost, where the default cost metric is execution time; however, users can specify alternative cost functions, such as monetary expense. Wayang emerges as a comprehensive and effective cross-platform data processing system tailored for unified data analytics. As of today, it supports various systems and libraries, including Spark, Flink, JDBC databases, Giraph, Tensorflow, and a in-memory Java-based executor.

We have recently extended Wayang in different directions. First, Wayang can now integrate big data systems and databases with machine learning (ML) systems. This creates an environment that allows an easy development of most modern pipelines that rely on data preprocessing, ML training, and prediction. We have further added a Python API to allow easy development for data scientists. Finally, we have added an SQL API which allows users to pose their queries using declarative language and can lead to integrating different database systems residing in different locations.

This demonstration showcases Wayang's capability to combine multiple data systems and data sources. We use a real use case comprising a data science pipeline for weather forecasting in Rio de Janeiro and well-known benchmarks to demonstrate how Wayang facilitates Extract-Load-Transform (ETL) processes without the need to centralize all data in one store to perform analytics.

2 Apache Wayang

Wayang integrates multiple data systems by providing an abstraction layer and a cross-platform optimizer. Figure 1 shows an overview. Users and applications can execute tasks via a Wayang job written in Java, Scala, or Python, or via a JSON REST API. Alternatively, they can use one of the libraries: the SQL library translates an SQL query to a Wayang job using Apache Calcite and the ML4all library provides an abstraction for easily developing ML algorithms [5].

Wayang's main building blocks are *data quanta* and *Wayang plans*. Data quanta is a fine-granular processing unit of data (e.g., row or line). A Wayang plan is an acyclic dataflow graph with loops where vertices are system-agnostic operators and edges are data flowing between two operators. Given a Wayang plan, the optimizer is responsible for determining the system on which each operator has to be executed, thus composing a system-specific *execution plan*. A user can also specifythe system on which each operator will run, bypassing the optimizer. The coordinator is then responsible for assigning the operators of the execution plan to the corresponding executor. The monitor checks whether the estimations used during the optimization are correct and if not, requests a new execution plan from the optimizer. We refer the reader to [2, 6–8] for more details on Wayang's architecture, cross-platform query optimizer, and data movement. In the following, we describe two new features that we have added, support for ML systems and a Python API.

2.1 ML Systems Integration

We have introduced the concept of a `Model` and implemented a new executor for Tensorflow. We followed Wayang's abstraction philosophy: We created an interface of a `Model` to be used by Wayang operators and then extended it for system-specific operators. We add the desired Wayang (system-agnostic) training operators, such as `LinearRegressionOperator`, which are unary operators that output a `Model`. Additionally, we create a `PredictOperator`, a `BinaryToUnary` operator which takes as input the data quanta and a model and outputs the data quanta with the predictions output by the model. System-specific execution operators, such as `SparkLinearRegressionOperator`, can be easily added as any other execution operator: extending the corresponding Wayang operator and providing the mappings from the Wayang to the execution operator. Additionally, a system-specific model needs to be instantiated (e.g., `SparkMLModel`) to be used as the output of the training operators and as input for the inference operator.

Unlike traditional machine learning models, the definition of deep learning models is more flexible. Users can combine different blocks (e.g., fully connected blocks, convolutional blocks) to build their desired models. The whole model can be represented as a graph on which the vertices represent blocks and the edges represent connections between blocks. In this case, we built a `DLModel` class that implements the `Model` interface, which contains a system-agnostic graph of the model defined by the user. For training, we implemented the system-agnostic `DLModelTrainingOperator`.

We have added Tensorflow as a new system by implementing a Tensorflow executor. The `TensorflowExecutor` is responsible for creating and destroying Tensorflow resources, such as a model graph and a model parameter context. When a training task is scheduled, it is mapped to `TensorflowDLModelTrainingOperator`. In this process, `DLModel` is converted to a `TensorflowModel`, which means that the user-defined model graph will be converted to a Tensorflow model graph. Likewise, for inference, `PredictOperator` is mapped to `TensorflowPredictOperator`. Using the `Model` abstraction, we have also added ML-related execution operators for Spark using Spark's ML library.

2.2 Python API

Adding a Python API has been an important milestone for Wayang as Python is the main programming language used nowadays by

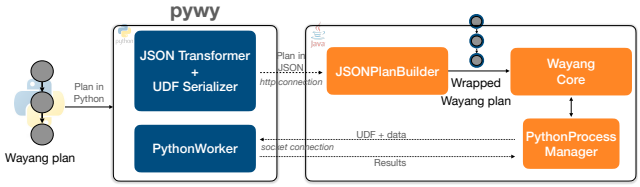


Figure 2: Python API workflow

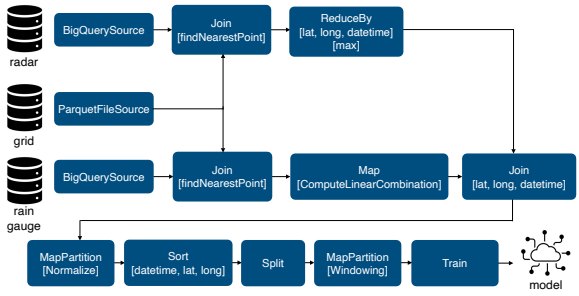


Figure 3: Weather forecast training pipeline

data scientists for building ML and AI applications. To achieve this, we had to overcome the challenge of integrating Python with the Java-based Wayang codebase. Although there are multiple tools integrating these two programming languages, the main challenge comes from the User Defined Functions (UDFs) encountered in most of Wayang's operators. We opted for a solution that follows the extensibility mindset of the project and can be used in the future for any other programming language API addition.

Figure 2 illustrates our approach. First, the `pywy` module transforms the Python program containing the Wayang plan into JSON and `pywy` serializes the UDFs into a binary format using pickle. The plan is then sent to Wayang's REST API, where the `JSONPlanBuilder` transforms it into a Java Wayang plan where operators forming a pipeline are *wrapped* with a `MapPartition` operator, to reduce communication overhead between Python and Java, and contain the serialized UDFs. As Python runtime is required for the execution of the UDFs, the `PythonProcessManager` interacts via a socket connection with a Python program (`Worker`) which is responsible for executing the corresponding UDFs. The `PythonWorker` receives the UDF and data and returns the results.

3 Demonstration

We will demonstrate Wayang using a real-world use case, in addition to commonly used datasets and workloads.

3.1 Real Use Case: Weather Forecast Training

This use case has been developed in the context of the *Rionowcast*[1] project [9]. The project aims to support Rio de Janeiro's monitoring services through extreme weather forecasts, using deep learning models and a combination of radar and precipitation data for model training and inference. The radar data source produces a dataset with schema `Radar(Latitude, Longitude, Datetime, Reflectivity)`. The precipitation data source includes data from rain gauges and weather stations. It produces a dataset with schema `Precipitation(Latitude, Longitude, Datetime, humidity, pressure, wind_direction,rainfall_hour, rainfall_6hour)`. Recordings on both data sources are generated at a 5-minute frequency. They are captured and stored in cloud Big Query tables. In addition to the two data sources, the pipeline accesses a local datalake dataset, `Grid(id,latitude, longitude)`, which holds a regular grid covering the city of Rio de Janeiro, with spatial coordinates indicating the positions where the rainfall volume is to be forecasted. The pipeline preprocesses the three datasets, aligning their spatiotemporal resolutions according to the coordinates at the Grid dataset, a regular 2D spatial grid of approximately 3kms distance between cells, covering a rectangular area of 17x27 km, and a temporal resolution of one hour. The merged dataset is then normalized, divided into temporal windows of $size = 6$ hours, and split into training and test datasets. The training dataset is used to build a ConvLSTM model, which forecasts the precipitation volume at each grid coordinate for three time steps. The Weather Forecast pipeline is depicted in Figure 3.

Using the above pipeline, we will demonstrate Wayang's ability to choose the best platform for each operator. Notably, a single platform cannot execute this pipeline as none of the available platforms can perform all the operations. Thus, Wayang will employ BigQuery to load the data, Flink or Java for the nearest neighbor join and preprocessing, and Tensorflow for training. Wayang is used for both testing with a small dataset but also for building the model that will be later deployed. The audience will be able to modify the pipeline via a Jupyter Notebook and/or specify different platforms per operator to witness the difference in performance.

3.2 Demonstration Scenarios

Besides the aforementioned real-world use case, we will employ the TPC-H[2] and JOB[3] benchmarks, along with Wikipedia as a large publicly available text corpus, to demonstrate the advantages of Wayang across various scenarios

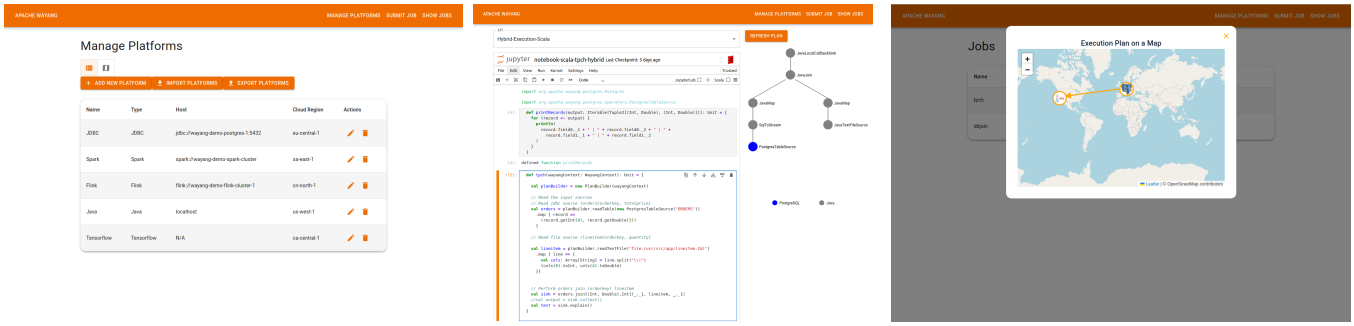
System Independence for Easy System Migration: We will showcase Wayang's system independence using a WordCount pipeline applied to a large text corpus from Wikipedia. This demonstration will illustrate how an application initially implemented as a standalone Java program–insufficient for processing large-scale data–can be seamlessly migrated to a big data system, such as Spark, by modifying a single line of code. This capability enables platform-agnostic application development, allowing developers to switch between different execution environments effortlessly without modifying the core application logic.

Cost-efficient ETL process: In this scenario, we will utilize diverse data sources to store different tables from the TPC-H and JOB benchmarks. For instance, some tables will reside in PostgreSQL, others in cloud or local filesystem-based blob storage, and some in cloud databases. This setup reflects a common enterprise environment where data is distributed across multiple storage mediums and locations. Traditionally, organizations must extract and centralize all data to perform analytics and generate visualizations. We will demonstrate how Wayang enables analytics directly over disparate data sources, eliminating the need for data duplication

[1] https://rionowcast.dexl.lncc.br/

[2] https://www.tpc.org/tpch/default5.asp

[3] https://github.com/gregrahn/join-order-benchmark



(a) Adding platforms (b) Writing jobs and visualizing their plans (c) Comparing and visualizing different jobs

Figure 4: Wayang's GUI

and additional processing infrastructure. This not only reduces data transfer and storage costs but also minimizes processing overhead.

Performance Speedup via the Optimizer: Finally, we will highlight Wayang's query optimizer, which constructs execution plans leveraging multiple data systems to minimize query runtime. In this interactive demonstration, participants will execute SQL queries on the TPC-H and JOB datasets through Wayang's SQL interface. The graphical user interface (GUI) will then provide a visual representation of the generated execution plans, allowing users to monitor performance improvements achieved through optimization.

3.3 User Experience

Our GUI will allow attendees to interact with Wayang in three ways: (i) users can manage data systems, (ii) users can write data processing tasks through different APIs, and (iii) users can explore the Wayang plan and the optimized execution plan.

System Management: As a first step, we demonstrate how easy it is to add the desired platforms to a Wayang installation (Figure 4(a)). For example, to add JDBC platforms (e.g., Postgres or MySQL), users enter the connection details (i.e., host, username, password), while to add Spark platforms, users enter the Spark master URL. We will provide predefined platforms for the demonstration but encourage the audience to connect their own. The audience can view the added platforms in an embedded map view.

Wayang APIs: Additionally, we will showcase Wayang's expressive APIs. Therefore, we embed Jupyter Notebooks into our GUI, which allows users to submit Wayang jobs through its Java, Scala, Python, and SQL APIs. After choosing the API where users want to express their job, they can formulate their Wayang job into the Jupyter Notebook. For demonstration purposes, we will provide predefined Wayang jobs, including the two use cases mentioned above, but we will also encourage the audience to implement their jobs. Users can use the newly added `explain()` functionality of Wayang to interactively view the produced plans (Figure 4(b)).

Wayang Optimizer: Further, we will showcase Wayang's optimizer capabilities. In particular, after submitting a job, users get a visual representation of the job in Wayang's intermediate representation (the Wayang Plan), a DAG consisting of platform-agnostic Wayang operators. Furthermore, users can see a visual representation of the inflated and the execution plan, which shows how the Wayang job will be executed. For example, for a job that applies a Java UDF and a selection over a PostgreSQL table, the Wayang plan will consist of a Map and a Selection operator, while the execution plan will consist of a Spark Map operator and a JDBC SQL statement with the selection. The audience can compare plans and their respective runtimes through this frame. Plans are also visualized in a map view, to illustrate Wayang's multi-cloud capabilities (see screenshot in Figure 4(c)).

User-based System Selection : Finally, users will be able to modify pre-existing pipelines and specify which system shall be used for specific operators by extending an operator in the pipeline with a function call (e.g., `.withTargetPlatform(Java.platform())`. Once the job is executed, users can again get a visual representation of the execution plan they constructed and monitor its performance.

Acknowledgments

The work is supported by the Carlsberg Foundation (CF-23-0836), the National Natural Science Foundation of China (No. 62272168), the Brazilian CNPq funding agency (No. 312100/2021-3), and the German Federal Ministry of Education and Research (BIFOLD24B and 01IS17052 as Software Campus project PolyDB).

References

[1] Divy Agrawal, Bertty Corteras-Rojas, Sanjay Chawla, Ahmed Elmagarmid, Yasser Idris, Zoi Kaoudi, Sebastian Kruse, Ji Lucas, Essam Mansour, Mourad Ouzzani, Paolo Papotti, Jorge-Arnulfo Quiané-Ruiz, Nan Tang, Saravanan Thirumuruganathan, and Anis Troudi. 2018. Rheem: Enabling Cross-Platform Data Processing – May The Big Data Be With You! *PVLDB* 11, 11 (2018), 1414–1427.

[2] Kaustubh Beedkar, Bertty Contreras-Rojas, Haralampos Gavriilidis, Zoi Kaoudi, Volker Markl, Rodrigo Pardo-Meza, and Jorge-Arnulfo Quiané-Ruiz. 2023. Apache Wayang: A Unified Data Analytics Framework. *SIGMOD Record* 52, 3 (2023), 30–35.

[3] Zoi Kaoudi and Jorge-Arnulfo Quiané-Ruiz. 2018. Cross-Platform Data Processing: Use Cases and Challenges. In *ICDE*. 1723–1726.

[4] Zoi Kaoudi and Jorge-Arnulfo Quiané-Ruiz. 2022. Unified Data Analytics: State-of-the-art and Open Problems. *Proc. VLDB Endow.* 15, 12 (2022), 3778–3781.

[5] Zoi Kaoudi, Jorge-Arnulfo Quiané-Ruiz, Saravanan Thirumuruganathan, Sanjay Chawla, and Divy Agrawal. 2017. A Cost-based Optimizer for Gradient Descent Optimization. In *SIGMOD*. 977–992.

[6] Zoi Kaoudi, Jorge-Arnulfo Quiané-Ruiz, Bertty Contreras-Rojas, Rodrigo Pardo-Meza, Anis Troudi, and Sanjay Chawla. 2020. ML-based Cross-Platform Query Optimization. In *ICDE*. 1489–1500.

[7] Sebastian Kruse, Zoi Kaoudi, Bertty Contreras-Rojas, Sanjay Chawla, Felix Naumann, and Jorge-Arnulfo Quiané-Ruiz. 2020. RHEEMix in the data jungle: a cost-based optimizer for cross-platform systems. *VLDB J.* 29, 6 (2020), 1287–1310.

[8] Sebastian Kruse, Zoi Kaoudi, Jorge-Arnulfo Quiané-Ruiz, Sanjay Chawla, Felix Naumann, and Bertty Contreras-Rojas. 2019. Optimizing Cross-platform Data Movement. In *ICDE*. 1642–1645.

[9] Fábio Porto, Mariza Ferro, and Eduardo S. Ogasawara et al. 2022. Machine Learning Approaches to Extreme Weather Events Forecast in Urban Areas: Challenges and Initial Results. *Supercomput. Front. Innov.* 9, 1 (2022), 49–73.