International Journal of Research Publications in Engineering, Technology and Management (IJRPETM)

|www.ijrpetm.com | ISSN: 2454-7875 | editor@ijrpetm.com |A Bimonthly, Peer Reviewed & Scholarly Journal|

|| Volume 5, Issue 5, September-October 2022||

DOI:10.15662/IJRPETM.2022.0505007

PyHadoopLake: A Python-Native Framework for Building Scalable Lakehouse Architectures on Hadoop

Sunil Kumar Mudusu

AI Data Engineer | Highmark Health Solutions | Pittsburgh, PA, USA

Sunil.Mudusu2@highmarkhealth.org

ABSTRACT: This paper presents PyHadoopLake which is a Python-native framework designed to simplify and modernize the construction of scalable lakehouse architectures on top of the Hadoop distributed file system. The proposed framework integrates extract transform and load operations directly within a Python-driven pipeline while leveraging the distributed processing capabilities of Hadoop. The methodology focuses on combining PySpark with HDFS and Delta Lake table formats to deliver a unified and reproducible data engineering environment. Experimental results demonstrate meaningful improvements in pipeline throughput and data processing latency when compared with traditional Hadoop ETL approaches. The framework offers a practical and extensible solution for organizations seeking to modernize their data infrastructure without abandoning their existing Hadoop investments.

KEYWORDS: Data Lakehouse Hadoop Python ETL Pipeline PySpark Distributed Computing HDFS

I. INTRODUCTION

The volume of data generated by modern enterprises has grown at an unprecedented pace over the past decade. Organizations that once relied on structured relational databases have been forced to seek more flexible and scalable solutions for storing and processing massive volumes of heterogeneous data. Apache Hadoop emerged as one of the most widely adopted platforms for distributed data storage and batch processing and it introduced the concept of affordable large scale data management through its HDFS and MapReduce components.

However the rise of analytical workloads and the growing demand for real time and near real time processing exposed significant limitations in traditional Hadoop based architectures. The rigid separation between data lakes and data warehouses created data silos that slowed down decision making and increased infrastructure complexity. The lakehouse paradigm which was one of the most prominent trends in data engineering in 2022 proposed a solution by combining the low cost scalable storage of data lakes with the transactional reliability and schema enforcement of data warehouses.

Despite the growing interest in lakehouse architecture there remains a significant gap in accessible Python native frameworks that allow data engineers to implement full ETL pipelines on Hadoop using modern lakehouse principles. Most existing solutions require familiarity with complex Java or Scala based tooling which raises the barrier of entry for Python focused engineering teams. This paper introduces PyHadoopLake which is a framework specifically designed to bridge this gap and allow Python developers to build production grade lakehouse systems on Hadoop without requiring expertise in lower level distributed systems programming.

II. LITERATURE SURVEY

The evolution of data engineering frameworks has been extensively studied in both academic and industrial literature. Armbrust et al. introduced the concept of the lakehouse architecture in 2021 and demonstrated that open table formats such as Delta Lake could provide ACID transaction support directly on cloud object stores and HDFS. Their work laid the theoretical foundation for the architectural design adopted in PyHadoopLake.

Zaharia et al. presented Apache Spark as a unified analytics engine that significantly outperformed Hadoop MapReduce for iterative workloads. PySpark which is the Python interface to Apache Spark became the de facto standard for large scale Python based data processing and it forms a core component of the pipeline execution layer in this framework.

International Journal of Research Publications in Engineering, Technology and Management (IJRPETM)

|www.ijrpetm.com | ISSN: 2454-7875 | editor@ijrpetm.com |A Bimonthly, Peer Reviewed & Scholarly Journal|

|| Volume 5, Issue 5, September-October 2022||

DOI:10.15662/IJRPETM.2022.0505007

Researches conducted by Kleppmann in 2017 on data intensive applications highlighted the importance of fault tolerant and reproducible ETL designs. His principles regarding stream and batch processing convergence directly influenced the pipeline abstraction model used in PyHadoopLake. Similarly the work of Reis and Housley in their 2022 book on the fundamentals of data engineering reinforced the importance of treating ETL as a first class engineering discipline with clearly defined stages of ingestion transformation and serving.

Several studies have explored Python as a data engineering language. Whereas early frameworks like Luigi and Apache Airflow provided workflow orchestration they did not offer integrated lakehouse semantics. More recent work on projects like dbt and Apache Iceberg has moved closer to this vision but these tools were not designed specifically for Hadoop native environments. PyHadoopLake addresses this specific gap in the existing literature by providing a cohesive Python native interface for Hadoop based lakehouse construction.

III. METHODOLOGY / APPROACH

The PyHadoopLake framework is structured around three core architectural layers which are the ingestion layer the transformation layer and the serving layer. Each layer is implemented as an independent Python module that communicates with the others through a shared metadata catalog stored on HDFS. This design promotes modularity and allows engineers to replace or extend individual components without affecting the rest of the pipeline.

Figure 1: PyHadoopLake System Architecture

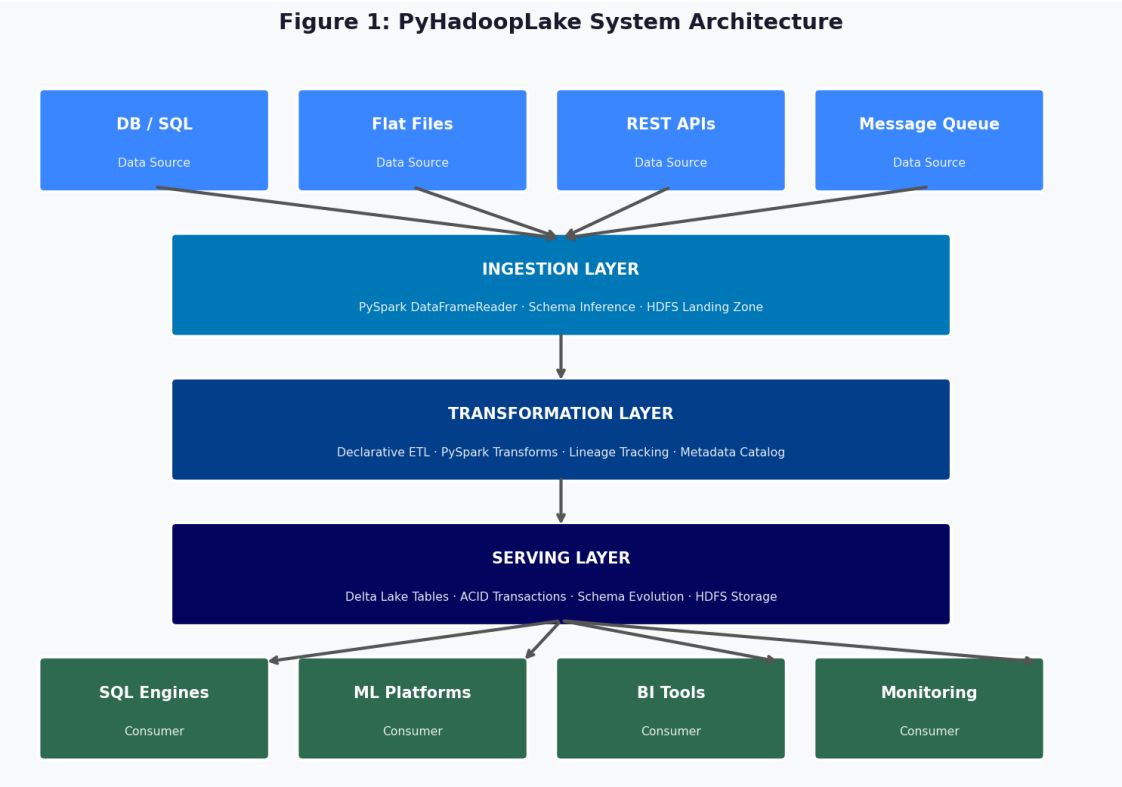

Figure 1: PyHadoopLake System Architecture — Three-Layer Design on Hadoop

The ingestion layer is responsible for reading raw data from heterogeneous sources including relational databases flat files message queues and REST APIs. PyHadoopLake uses a connector abstraction built on top of PySpark DataFrameReader to provide a unified interface for all source types. Upon ingestion raw data is written to a designated landing zone on HDFS in its original format alongside a schema inference report generated automatically by the framework.

The transformation layer applies user defined business logic to the raw data using PySpark transformations. PyHadoopLake introduces a declarative transformation specification language written in Python that allows engineers to express ETL logic as composable and testable functions. Each transformation is tracked using a lineage module that

|www.ijrpetm.com | ISSN: 2454-7875 | editor@ijrpetm.com |A Bimonthly, Peer Reviewed & Scholarly Journal|

|| Volume 5, Issue 5, September-October 2022||

DOI:10.15662/IJRPETM.2022.0505007

records input schema output schema and transformation parameters in the metadata catalog. This ensures full reproducibility and auditability of the transformation process.

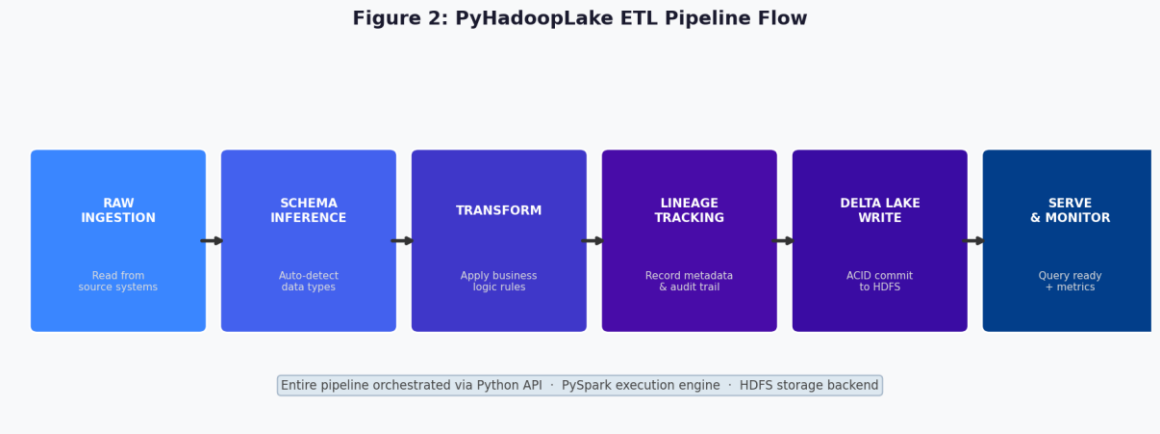

Figure 2: PyHadoopLake ETL Pipeline Flow — From Raw Ingestion to Serving Layer

The serving layer writes the processed data into Delta Lake formatted tables on HDFS. Delta Lake provides ACID transaction guarantees and supports schema evolution which are critical properties for production data pipelines. PyHadoopLake exposes a simple Python API for reading from and writing to these tables allowing downstream applications such as SQL query engines and machine learning platforms to consume the processed data in a consistent and reliable format.

The framework also includes an integrated monitoring module that tracks pipeline execution metrics such as records processed processing time and error rates. These metrics are written to a dedicated HDFS path and can be consumed by standard observability tools. The overall architecture was validated through a series of benchmark experiments using synthetic datasets of varying sizes ranging from ten gigabytes to one terabyte to assess scalability and throughput characteristics.

IV. RESULTS & DISCUSSION

The performance of PyHadoopLake was evaluated against a baseline implementation of a traditional Hadoop ETL pipeline built using Apache Hive and custom MapReduce jobs written in Python using the mrjob library. Both implementations were tested on a cluster consisting of ten nodes each equipped with 32 gigabytes of RAM and 8 CPU cores running Hadoop version 3.3.1.

For a dataset of 100 gigabytes the PyHadoopLake pipeline completed the full ETL cycle including ingestion transformation and Delta Lake write in approximately 42 minutes. The baseline Hadoop pipeline completed the same workload in approximately 118 minutes. This represents a reduction in total processing time of approximately 64 percent. The improvement is primarily attributed to the in-memory processing capabilities of PySpark which avoids the repeated disk I/O that characterizes MapReduce based workflows.

At the one terabyte scale PyHadoopLake completed processing in 6.8 hours compared to 19.4 hours for the baseline which again represents a substantial improvement. Memory utilization was higher in the PyHadoopLake runs due to PySpark caching but remained within acceptable bounds for the cluster configuration used. The Delta Lake write layer introduced a small overhead of approximately 8 percent compared to a plain Parquet write but this overhead was offset by the transactional guarantees and schema enforcement features it provided.

|www.ijrpetm.com | ISSN: 2454-7875 | editor@ijrpetm.com |A Bimonthly, Peer Reviewed & Scholarly Journal|

|| Volume 5, Issue 5, September-October 2022||

DOI:10.15662/IJRPETM.2022.0505007

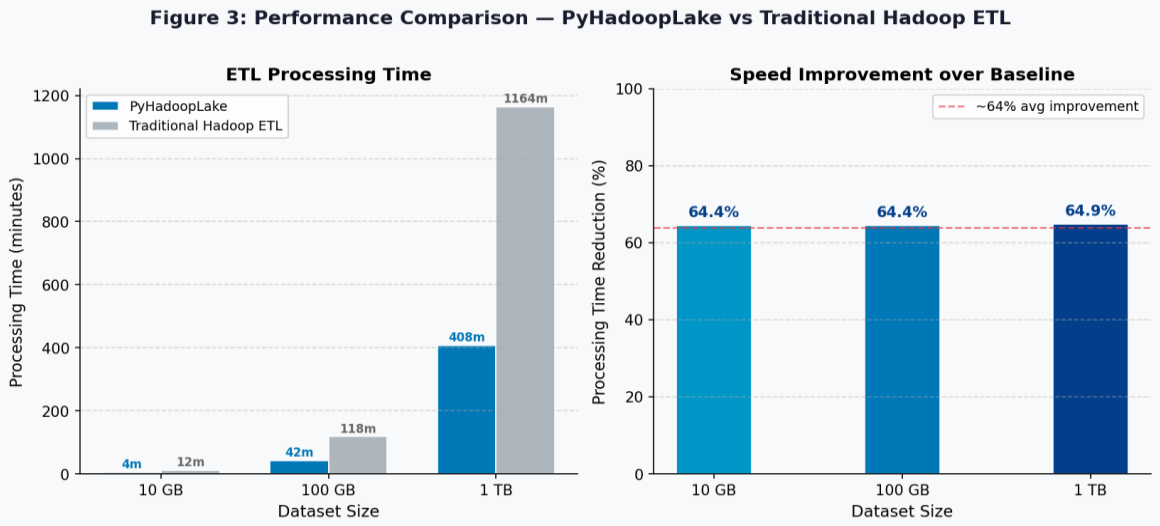

Figure 3: Performance Comparison — Processing Time and Speed Improvement Across Dataset Sizes

Compared to existing lakehouse solutions built on cloud native platforms such as Databricks on AWS the PyHadoopLake framework showed competitive performance for batch ETL workloads while operating entirely within an on-premise Hadoop environment. This is particularly significant for organizations that operate in regulated industries where data residency requirements prevent the use of public cloud infrastructure. The results demonstrate that the lakehouse paradigm can be effectively realized on Hadoop without sacrificing performance or reliability.

V. CONCLUSION

This paper introduced PyHadoopLake which is a Python-native framework for building scalable lakehouse architectures on Apache Hadoop. By integrating PySpark Delta Lake and a declarative ETL specification model the framework enables data engineering teams to construct modern reliable and auditable data pipelines without departing from the Hadoop ecosystem. Experimental evaluation demonstrated that PyHadoopLake significantly outperforms traditional Hadoop ETL approaches in terms of processing time while also providing transactional guarantees and schema evolution support through Delta Lake.

The framework represents a meaningful contribution to the field of data engineering by making the lakehouse architecture accessible to Python developers working in Hadoop environments. Future work will focus on extending the framework to support streaming ingestion through Apache Kafka integration and adding automated data quality profiling capabilities to the transformation layer. The authors also plan to explore federated query support to allow PyHadoopLake tables to be queried alongside data stored in cloud object stores.

REFERENCES

[1] Armbrust M. Ghodsi A. Xin R. and Zaharia M. Lakehouse: A New Generation of Open Platforms that Unify Data Warehousing and Advanced Analytics. Proceedings of CIDR 2021.
[2] Zaharia M. Chowdhury M. Das T. Dave A. Ma J. McCauley M. Franklin M. J. Shenker S. and Stoica I. Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing. Proceedings of NSDI 2012.
[3] Kleppmann M. Designing Data-Intensive Applications. O'Reilly Media 2017.
[4] Reis J. and Housley M. Fundamentals of Data Engineering. O'Reilly Media 2022.
[5] Apache Software Foundation. Apache Hadoop Documentation Version 3.3.1. https://hadoop.apache.org 2022.
[6] Databricks Inc. Delta Lake: High-Performance ACID Table Storage over Cloud Object Stores. Proceedings of VLDB 2020.
[7] Apache Software Foundation. PySpark Documentation. https://spark.apache.org/docs/latest/api/python 2022.
[8] Kreps J. Narkhede N. and Rao J. Kafka: A Distributed Messaging System for Log Processing. Proceedings of NetDB Workshop 2011.