Security of Infocommunication Systems and Internet of Things

2025 Vol 3, No 1

https://doi.org/10.31861/sisiot2025.1.01004

Received 14 May 2025; revised 07 June 2025; accepted 08 June 2025; published 30 June 2025

Information Technology for Data Compression and Transformation by Means of Amazon EMR

Yevhen Kyrychenko[1,*] and Igor Malyk[2]

[1]Software Engineering Department, Yuriy Fedkovich Chernivtsi National University, Chernivtsi, Ukraine
[2]Department of Mathematical Problems of Control and Cybernetics, Yuriy Fedkovych Chernivtsi National University, Chernivtsi, Ukraine

*Corresponding author (E-mail: kyrychenko.yevhen@chnu.edu.ua)

ABSTRACT As data processing volumes grow in various fields, the demand for applications capable of efficiently managing, processing, and transforming large amounts of information is also increasing. Modern approaches to storing and processing large amounts of data are primarily based on universal text formats, such as CSV and JSON. Their prevalence can be explained by their compatibility with a wide range of software tools and ease of integration. These formats are inefficient when dealing with massive volumes of data, particularly when scaling systems or executing analytical queries. The lack of built-in compression, row structure, and metadata leads to significant time and computing resources, which creates a conflict between the requirements for speed and cost-effectiveness of processing and the technical capabilities of traditional text formats. Columnar storage formats, such as Parquet and ORC, offer an alternative. They employ a compact structure tailored for quick analytical queries in distributed computing settings. Effective coding, indexing, and built-in compression techniques considerably lower data sizes and speed up processing. This research aims to develop and experimentally verify the technology of automated data conversion from inefficient text formats to Parquet and ORC formats using Apache Airflow and Amazon EMR. The proposed architecture involves creating a cloud pipeline that performs data conversion and subsequent storage in formats focused on analytical workloads. The system uses Apache Airflow for process orchestration, Amazon EMR and Apache Spark for distributed processing, AWS S3 as scalable storage, AWS Glue for metadata management, and Amazon Athena for SQL access to transformed data. This approach solves performance problems by offering a flexible, reliable, cost-effective solution that adapts to different work scenarios and workloads.

KEYWORDS information technology, Big Data, AWS, distributed processing, data compression.

I. INTRODUCTION

As modern approaches to storing and processing large amounts of data are mostly based on universal text formats, such as CSV and JSON. These formats are popular because they integrate easily and work well with a wide range of software tools. However, when dealing with large datasets, their efficiency drops especially for analytical queries or when scaling systems. Without built-in compression and with data arranged in a way that's not optimized for column-based access, they end up consuming far more time and computing resources than necessary. Thus, a contradiction arises between the growing requirements for the speed and efficiency of information processing and the technical capabilities of traditional text formats.

The purpose of this study is to develop and experimentally verify the technology of automated data conversion from inefficient text formats to columnar storage formats, in particular Parquet and ORC, using Apache Airflow and Amazon Elastic Map Reduce (EMR). It is assumed that such a solution will significantly increase the performance of analytical queries, as well as reduce the cost of storing and processing information in the cloud environment.

This research examines the main drawbacks of CSV and JSON formats when handling large-scale data and explores the advantages of columnar formats designed for analytical workloads. Building on this analysis, we created a cloud-based pipeline that automatically transforms data and saves it in Parquet and ORC formats. It operates within the AWS environment, with Apache Airflow handling workflow orchestration and Amazon EMR managing the distributed data processing.

As the amount of data that companies process grows, there is a demand for tools that can handle and process large data sets. CSV and JSON formats are the most popular and commonly used data formats. However, they fail to provide usable and efficient data representation for scalable analysis, as they do not compress, have a string-based format and provide no built-in metadata. Instead, Parquet and ORC shall be compact, columnar, and optimized for analytical query performance in a distributed computing environment [1, 2]. Here, we report the development of a cloud-based pipeline intended to facilitate the conversion of raw data to these alternative, more analytically friendly representations. This system is built on top of state-of-the-art technologies such as Apache Airflow for orchestration [3], Amazon EMR with Apache Spark for distributed processing [4], AWS S3 for scalable storage [5], AWS Glue for maintaining metadata [6], and Amazon Athena for querying transformed data using SQL [7]. This implementation provides a performance solution that is both cost-effective and scalable, that can be easily monitored, and is also flexible across a multitude of workloads [8].

II. SYSTEM ARCHITECTURE

A cloud-based pipeline was created for data processing, which is easily scalable and works almost without human intervention. Each component performs its straightforward task – it is responsible for managing, launching, and optimizing the entire process. As shown in Figure 1, the whole architecture combines AWS services and open source tools, making the system stable, efficient, and easy to use.

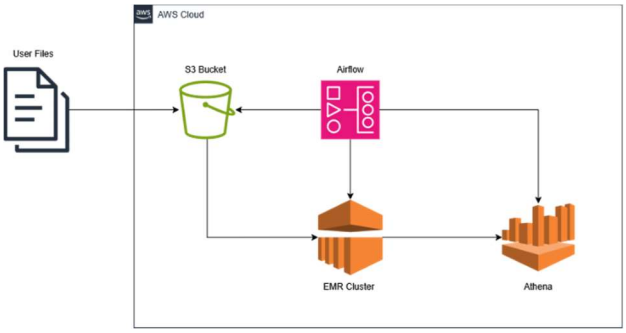

FIG. 1. System Architecture.

A. Data Acquisition and Storage. It starts with storing raw files – usually CSV or JSON – in Amazon S3. This storage acts as a kind of "gateway" for the data and as a buffer before further processing. To ensure that the pipeline runs automatically and without failures, the folder structure in S3 is designed so that the names immediately show essential information: where the data came from, when it arrived, and what it contains. This allows the system to "understand" what it is dealing with and run the right processing logic.

We chose Amazon S3 because it is stable, easily scalable, and integrates well with other AWS services. In addition, it has such useful features as file versioning, flexible access control, and object change events – all of which are very helpful in tracking where the data came from and ensuring stable and repeatable processing [5].

B. Process Orchestration with Apache Airflow. After data is placed in Amazon S3, the orchestration layer implemented with Apache Airflow monitors the appearance of new files in the storage. A Directed Acyclic Graph (DAG) is activated when a new object is detected, initiating a specific processing task sequence. Each DAG execution is configured with the specified parameters, allowing the workflow to dynamically adapt to various data formats and sources [3].

When a new file is detected, Airflow starts a transformation process that creates an EMR cluster, launches a Spark application, and logs the results. Airflow has a flexible architecture, so it can automatically repeat tasks in case of errors, run multiple tasks simultaneously, and send notifications in real time. It gives you complete control over how data processing is going. All important details – such as task statuses and logs – are stored in its metadata, so monitoring and auditing the system is relatively easy.

This orchestration allows you to process data quickly and efficiently. It also scales easily, allowing multiple workflows to run in parallel for different data sources.

C. Distributed Processing with Spark on Amazon EMR. The pipeline's backbone is the transformation logic developed using PySpark running on an Amazon EMR cluster. EMR is a managed service for Hadoop and Spark that enables scalable and fault-tolerant processing of large data sets [9]. Using Spot Instances and auto-scaling, the system reduces costs while maintaining performance [10].

PySpark tasks:

- Read raw input data from S3;
- Clean and normalize data (remove empty lines, address missing values, remove unwanted characters);
- Determine variable types (e.g., numeric, categorical, boolean, text, timestamp);
- Generate concise representations (e.g., statistical summaries, frequency tables, TF-IDF vectors);
- Write transformed data to columnar formats.

The power of Spark allows the system to process terabytes of data efficiently, executing tasks in parallel on multiple nodes. Intermediate transformations are cached in memory when necessary to improve execution efficiency.

D. Compact Storage in Parquet and ORC. After transformation, the resulting datasets are serialized into columnar formats, such as Apache Parquet or ORC, and stored in a designated source storage on S3. These formats were chosen because they:

- Offer columnar storage that allows for fast and selective reads;
- Include built-in compression (Snappy, ZSTD).

Storing data in Parquet or ORC formats reduces file size and speeds up analytical queries. To make it easier to navigate the data, we use a standardized file and folder naming scheme – they encode where the data came from, when it was processed, and which version of the schema was used. It greatly facilitates control, auditing, and maintaining cleanliness in the system.

E. Metadata Integration with AWS Glue and Query Access with Athena. Each transformed dataset is automatically registered in the AWS Glue metadata catalogue to simplify queries and integrate with analytics tools. Glue acts as a centralized repository that stores information about schemas, partitions, and physical locations of files. Once registered, the data becomes available for instant querying through Amazon Athena, a serverless SQL engine that operates directly on data in S3 using standard SQL syntax. This approach avoids additional ETL steps or deploying a traditional DBMS. As a result, analysts, data scientists, and other services can quickly and cost-effectively access stored data.

III. DATA PROCESSING WORKFLOW

The data processing process is at the heart of the pipeline, which transforms raw files into clean, structured, and query-ready formats. This process includes validating the data, standardizing it, adding valuable context, and transforming it into efficient columnar formats. The entire module was implemented as a distributed PySpark application and is fully managed by Apache Airflow. It is built to handle a wide range of data sources and is designed for scalability and fault tolerance when running on Amazon EMR.

A. Raw Data Loading and Preprocessing. When a new data file is uploaded in Amazon S3, the DAG in Apache Airflow detects it and triggers an Amazon EMR cluster and delegates the processing tasks using PySpark. The first step is to load the file using Spark's built-in CSV or JSON readers optimized for distributed parsing. The system is able to handle correctly:

• files with or without headers;
• non-standard delimiters and encodings;
• missing or NULL values;
• partially corrupted or noisy records.

Data cleaning occurs immediately after loading and involves removing all extraneous whitespace and control characters, standardizing field delimiters, and replacing missing or incorrect values with predefined or statistically valid ones (e.g., median for numeric variables or mode for categorical variables).

This preprocessing step provides a consistent and clean foundation for subsequent analytical analysis and data type determination.

B. Variable Type Inference. A critical workflow component is the identification and classification of variable types. Each column in the dataset is individually examined using heuristic and statistical methods to determine its semantic type. The pipeline covers the following categories:

• Boolean: Binary values such as True/False, 1/0, or Yes/No;
• Factorial/Categorical: Columns that contain a limited set of unique values with low cardinality (e.g., product types, country codes);
• Numeric: Integers or floating-point numbers that represent measured quantities;
• Temporal: Strings or numbers that can be converted to standard date and time formats;
• Text/String: Free text that can be structured (e.g., product IDS) or unstructured (e.g., descriptions, comments).

The column type is determined based on several statistical characteristics, including the number of unique values, entropy, average string length, and the rate of successful casts to a particular type. For example, a column is classified as categorical if the number of unique values does not exceed 20 and the entropy remains low. Instead, a column is temporal if its values conform to known date and time formats and demonstrate regularity of intervals.

This classification is the basis for applying aggregation and vectorization methods, allowing you to preserve the meaningful data load while reducing its dimensionality.

C. Building a Compact Representation. Once the system has figured out the datatypes of the variables, it creates a compact but data-rich summary for each column within the dataset. These representations are chosen based on criteria to facilitate subsequent analysis and minimize data that should be stored, but the content is preserved:

• Boolean and Categorical: The count and proportion are calculated for each unique value to build the frequency distribution. It helps detect class imbalance or categorical skew;
• Numeric: The pipeline calculates statistics such as mean, standard deviation, minimum, maximum, and variance. In addition, percentiles or histograms can be created for skewed distributions;
• Temporal: Temporal variables are segmented into hourly, daily, or monthly intervals, and the resulting time series is converted to a histogram of events. Temporal patterns and anomalies can be found more easily with it;
• Text/String: The system uses TF-IDF (Term Frequency – Inverse Document Frequency) vectorization for unstructured text. It produces sparse feature vectors which highlight semantically important words in the collection. The TF-IDF matrix is limited, so it consumes less memory [11].

The output is a more complex and smaller original data structure suitable for machine learning, outlier detection and data quality evaluation tasks.

D. Compaction and Serialization. Once the compact views are ready, the next step is compressing and storing the data in an analytics-friendly columnar format. Apache Parquet or ORC are most commonly used for this, both of which have their advantages:

• There is built-in compression (such as Snappy or ZSTD) that reduces the size of the files;
• Complex data structures and types are supported;
• The formats are compatible with popular analytics tools such as Athena, Presto, or Redshift Spectrum.

All prepared files are stored in Amazon S3 in clearly structured folders. The file names and paths include information about where the data came from, when it was processed, and which schema version was used. This simplifies tracking change history and allows you to manage different versions easily. And if you need to work with large volumes, you can additionally partition the data, which significantly speeds up analytical queries. Additional partitioning can be applied to speed up analytical queries on extensive data.

IV. AUTOMATION AND ORCHESTRATION WITH APACHE AIRFLOW

Automation is a fundamental principle in modern data engineering pipelines. Within this framework, Apache Airflow functions as the primary orchestrator that manages the complete data processing lifecycle, from input discovery to final output logging. Airflow provides the planning, execution, and dependency management necessary to ensure consistent, reproducible, and scalable execution of data processing workflows.

A. The Role of Airflow in the Pipeline. Apache Airflow plays a crucial orchestration role in the given architecture by constantly observing the status of the specified Amazon S3 bucket looking for new raw files. By detecting one or more new entities a pre-defined DAG is automatically triggered that includes the full logic controlling how to start the transformation of the data. The DAG also serves to orchestrate the execution of tasks, to keep them ordered and to provide them with the full access to the input resources ever at their point of work [12].

There are several significant benefits to using airflow here:

• Decoupled logic: The different parts of the workflow (i.e., cluster creation, submission of the job, the metadata registration) were written as individual tasks;
• Retries and error handling: If an error occurs during a

task execution – for example, due to a short-term network outage or incorrect input – Airflow will automatically try to run it again, guided by the specified rules;

• Execution transparency: All the information of task status, execution time, logs, errors is kept inside Airflow's storage, so you can know the details of what's happening in the system.

This modular task structure leads to ease of maintenance, and the declarative DAG nature of the it enhances visibility and auditability.

B. DAG Execution Flow. A typical Airflow DAG in this pipeline consists of the following sequence of tasks:

1. S3 Sensor Task: Monitors the arrival of new files to a specified S3 path. It can be configured using regular expression filters, object size conditions, or time window specifications.

2. Cluster Provisioning Task: Initiates a temporary Amazon EMR cluster with configuration parameters, including instance type, point-to-point or on-demand options, autoscale policies, and upload actions.

3. Spark Job Submission Task: Submits a PySpark job to the EMR cluster, pointing to a script or application stored in S3. The job parameters can be dynamically generated based on metadata extracted from the file path or its contents.

4. Job Monitoring Task: Polls the EMR cluster to track the progress of the Spark job. Airflow can trigger an alert or escalate it to a backup engine if the job fails.

5. Validate results: The system verifies that the required source files appeared in S3, in the correct format, and where they should be.

6. Register with Glue: The API calls AWS Glue to register or update the schema and data storage location, allowing them to be conveniently queried through Amazon Athena.

7. Cluster Dismantle Task: Shuts down the EMR cluster to prevent unnecessary overhead if automatic shutdown was not configured during setup.

A DAG can also have conditional branching, parallel sub-DAGs, or cross-DAG triggers in more complex deployments, allowing for the orchestration of multiple datasets or related ETL stages.

C. Parameterization and Dynamic Configuration. Airflow's support for templates and environment variables allows the pipeline to be parametric and adaptable to different input characteristics or processing needs without changing the DAG logic. For example:

• A file path such as s3://raw-data/logs/2025/05/04/ can be parsed to obtain dataset metadata (e.g. date, domain);

• File-specific schemas, transformation instructions, or cluster size parameters can be injected at runtime;

• Multiple DAGs can be created from a single template to facilitate parallel processing different datasets or business units.

This design promotes reuse and reduces duplication.

D. Monitoring and Alerting. Another essential aspect of maintaining a stable pipeline in production is the operational visibility of the whole execution. Airflow provides adequate monitoring tools that can be accessed via the web interface or REST APIS provided or integrated with other services, such as Prometheus, Slack, or PagerDuty.

It possesses the following features: real-time visualization of the DAG execution, detailed logs of every task, including stdout and error streams, email or webhook notifications in achieving SLA or errors export metrics for the dashboards and monitoring systems, etc.

These tools allow the engineering teams to provide control over all the execution stages and react rapidly to any issue before it can influence the business processes, which ensures excellent reliability and controllability of the whole pipeline.

V. EVALUATION METHODOLOGY

The Evaluation Methodology was developed to measure the proposed data transformation pipeline's efficiency, performance, and cost-effectiveness. It includes measuring the primary metrics, such as the amount of data stored following the processing, the execution of transformation time, the processing of analytical queries when ready, and the financial cost for the computational resources.

This data was collected using a combination of empirical benchmarks, built-in monitoring and analytics tools from the AWS arsenal. The testing was conducted in conditions as close to real-world use as possible: with typical datasets, typical load processes, and configurations of Amazon cloud services.

A. Datasets and Experimental Configuration. The evaluation applied stock trading time series data sets obtained from open financial platforms [13]. The topic was trading information for the shares of Apple and Tesla over several trading sessions. These datasets were chosen as they were semi-structured, numeric measures and timestamps, and found to be most interesting for testing analytical queries. Each dataset was uploaded to Amazon S3 in CSV format and ran in the described pipeline above. The Spark-based transformation workflow was executed on an Amazon EMR cluster with the following configuration:

• Cluster type: EMR transient (auto-terminated);

• Node configuration: one primary node, 2-4 primary nodes (m5.xlarge), with auto-scaling enabled;

• Spark executor memory: 4 GB;

• File sizes: 100 MB – 1 GB for each dataset.

Output files were generated in Parquet and ORC formats for benchmarking. All executions were fully automated using Apache Airflow data groups. In multi-cloud deployments, the pipeline can integrate with systems like Cassandra for persistent NoSQL storage [14].

B. Key Evaluation Metrics. The following four categories of metrics were established to evaluate the system at both the functional and operational levels:

Storage Size Reduction. The original file size (CSV) ratio to the compressed Parquet or ORC file size was calculated to assess the compression efficiency. This metric is vital for understanding the long-term storage savings and the reduction in input/output in subsequent queries.

Compression Ratio = CSV_Size / Parquet_Size/ORC.

Transformation Latency. The transformation latency quantifies the time from file detection to completing the

PySpark job, including cluster startup, processing, and output recording. Latency was documented using:

• Airflow job logs and execution timestamps;

• CloudWatch EMR job monitoring (step start/stop time).

This metric is crucial for assessing the throughput of the pipeline and its applicability for near-real-time operations or daily batch operations.

Query Speed. To analyze the improvements in analytical performance, identical SQL queries were executed on raw CSV data and compressed Parquet/ORC data using Amazon Athena. The queries included:

• SELECT with WHERE filters based on numeric ranges;

• Aggregations (AVG, MAX, MIN);

• Slices and groupings based on dates.

The Athena query execution time and the amount of data scanned were recorded for each execution.

Cost Analysis. The pipeline's financial impact was monitored using the AWS Billing and Cost Explorer dashboards. It included:

• EMR Costs: Billing for cluster execution (EC2, EBS, and Spark overhead);

• S3 Storage: Before and after data compression;

• Athena Query Costs: Estimated based on data scanned for each query.

Where possible, cost optimization techniques (such as point instances and automatic cluster termination) were used and their impact was documented.

C. Metrics Collection Tools. Metrics data were collected from multiple sources to ensure accuracy and reproducibility:

• Metric Source;

• Storage Size S3 Object Metadata;

• Job Latency Airflow Logs, CloudWatch Metrics;

• Query Performance Athena Execution Logs [15];

• Cost and Usage AWS Billing Dashboard;

• Resource Utilisation EMR Metrics, Spark UI.

VI. RESULTS AND DISCUSSION

Practical testing showed that data is stored more efficiently, queries are executed much faster, and the system operates stably even with increased load. Such achievements indicate the practical reliability and feasibility of implementing the proposed architecture in environments close to real production, considering modern requirements for processing large volumes of data. The following sections present a detailed analysis of the results in four key areas: storage reduction, transformation latency, query performance, and financial efficiency.

A. Storage optimization. One of the most significant results of the applied transformation method was a significant reduction in data volumes by moving from raw CSV to columnar storage formats. As shown in Table 1, the transformation to Parquet and ORC formats allowed compression ratios of 3.5× to 5.2×, which varied according to the structure and level of redundancy in the initial data sets.

After analyzing the data we received, we see that Apple's daily dataset, which initially had a size of 412 MB in CSV format, was reduced to 87 MB in Parquet format and 79 MB in ORC format. Similarly, Tesla's dataset was reduced from 398 MB to 114 MB (Parquet) and 98 MB (ORC). In addition to optimizing disk space use, the storage size reduction decreased the volume of data transported during analytical query execution, which decreased expenses and improved processing performance.

TABLE 1. Compression Performance of Parquet and ORC.

Dataset	Format	Original Size (CSV)	Compressed Size
Apple (daily)	Parquet	412 MB	87 MB
Tesla (daily)	Parquet	398 MB	114 MB
Apple (daily)	ORC	412 MB	79 MB
Tesla (daily)	ORC	398 MB	98 MB

Both Parquet and ORC did a good job of shrinking the size of the data, although ORC was somewhat more successful because of its more advanced encoding techniques and integrated indexing. Both Parquet and ORC did a good job of lowering the volume of data, but because of its more advanced encoding techniques and integrated indexing, ORC was marginally more effective. It improved performance and decreased expenses by saving storage space and transferring less data during analytical queries.

B. Transformation latency. The time it took to fully process each dataset, from when the system detected it to when it completed compression, varied slightly depending on how quickly the clusters were started and how many resources were allocated. On average, during testing, the transformation latency was:

• Apple dataset (412 MB): ~5.2 minutes;

• Tesla dataset (398 MB): ~4.8 minutes.

Most of the execution latency was the time to deploy the Amazon EMR cluster, which averaged about two minutes. In contrast, the direct processing of the data using Spark took between 2.5 minutes and 3 minutes for each dataset. The results indicate the proposed pipeline's suitability for daily batch transformations or even more frequent updates in systems where real-time processing is not required.

Analysis of Airflow logs showed stable performance at the individual task level throughout all test runs. The automated retry mechanism effectively compensated for episodic delays in cluster initialization, which were recorded in one of the nine tests.

C. Improved query performance. SQL queries executed through Amazon Athena showed significantly improved speed and cost-effectiveness when working with columnar formats compared to regular CSV files.

TABLE 2. Formats for Apple and Tesla Datasets.

Query Type	CSV (ms)	Parquet (ms)	ORC (ms)
Filter by date range + aggregation	2350	580	490
Compute daily averages	1970	490	465
Top N by volume	3120	610	540

Switching to storing compressed output instead of raw input data provided an estimated 70% reduction in storage costs in Amazon S3. Additionally, because of the reduced amount of data to be scanned, query execution costs in

Amazon Athena decreased by approximately 65%. Implementing autoscaling and auto-termination of the cluster reduced the compute costs in the Amazon EMR environment to less than $0.25 per data set, even when using temporary clusters.

D. Reliability and Observability. During testing, the orchestration layer was implemented through Apache Airflow reliably handled job launches, error handling, and the entire DAG lifecycle. Airflow logs and metrics from CloudWatch provided detailed insight into how individual jobs performed and how efficiently resources were utilised [16].

Key operational observations include:

- No manual intervention was required during successful DAG execution;
- Automatic resolution of job failures (e.g., temporary S3 access issues) through Airflow retries;
- Cluster auto-termination function worked as expected, providing cost-effectiveness.

This observation confirm that the pipeline is well-suited for production environments where automation, error handling, and transparency are critical.

VII. PRACTICAL APPLICATION AND SCALABILITY

In addition to being good at transforming raw data into a format that is easy to analyze, this pipeline is also very flexible in its structure. It can be easily adapted to the needs of any field that deals with large amounts of data. Due to this versatility, the system is suitable not only for financial or logistics tasks, but also for medicine, retail and many other things. Such versatility makes the system useful and scalable for multiple application scenarios – from finance and logistics to healthcare or retail. Thanks to its modular design, cloud infrastructure orientation, and integration with popular opensource tools and native AWS services, the system provides easy implementation into corporate processes, compatibility with existing data processing platforms, and support for modern operational analytics practices.

A. Corporate data warehouse and analytics. The proposed pipeline can be critically important in building a modern data processing and storage architecture for organizations that regularly receive large volumes of logs, transactional data, or information from IoT devices. Converting raw CSV or JSON files to Parquet, or ORC columnar formats, reduces long-term storage costs. It significantly improves the efficiency of analytical queries in systems such as Amazon Athena, AWS Redshift Spectrum, or Spark SQL.

For example, financial companies that process large volumes of daily trading transactions can use this pipeline to automatically clean, compress, and log data to a centralized metadata catalogue. It enables analysts to perform near-real-time performance queries and historical analysis with minimal latency and without infrastructure expansion.

B. Data Quality Monitoring and Profiling. With built-in variable classification and aggregation capabilities, the pipeline also functions as a data profiling tool that can generate metadata-rich summary data for new datasets. These summaries, including frequency distributions, statistical moments, or TF-IDF vectors, can detect anomalies, validate input data streams, or track pattern changes. When supplemented with historical baseline or comparison features (e.g., using the DISS metric mentioned in the previous study), the system can act as a data quality monitoring utility that flags data structure or content inconsistencies, thereby supporting data governance efforts.

C. Transformation latency. The mechanisms for determining variable types and constructing a compact representation in this pipeline are similar to those used in typical preprocessing in machine learning tasks. Due to this similarity, the system can be easily integrated with ML pipelines and immediately receive ready-to-train, validated feature sets at the output [17].

With minor adjustments, the pipeline can be extended to include functions such as feature selection, encoding, normalization, or even label generation, which is fed into subsequent training processes in environments based on SageMaker, Databricks, or Kubernetes.

D. Real-time and streaming extensions. While the existing pipeline is tailored for batch processing, its modular structure allows future connections to real-time data sources. AWS-native services such as Amazon Kinesis, MSK (Managed Kafka), or AWS Lambda can run lightweight, low-latency transformations on incoming records, and compression and storage are managed asynchronously in micro-batches. It will enable streaming transformation of log data, click activity, IoT telemetry, or social media interactions, meeting the needs of applications for fraud detection, operational monitoring, or customer behavior analytics.

E. Multi-cloud and hybrid deployment. The current version is AWS-centric, but the tech is agnostic (Airflow and Spark). With minimal configuration, the pipeline can also be deployed to Google Cloud Dataproc or Azure Synapse! - or to an on-premises Spark cluster. It can be easily orchestrated by Airflow (running on Kubernetes) or linked with CI/CD for degrees of automation, flexibility, etc. This flexibility results in portability across any environment, which is particularly useful for organizations that leverage hybrid or multi-cloud infrastructures. The consequence is less lock-in to one cloud provider and more freedom in configuration.

VIII. CONCLUSION

This paper presents a cloud-native data transformation pipeline that can transform raw formats such as CSV and JSON into high-performance columnar structures, such as Parquet and ORC, in a scalable and automated manner. With a modular architecture and management via Apache Airflow, the system combines distributed processing on Amazon EMR with robust metadata management and data access via AWS Glue and Amazon Athena. The framework's design is driven by goals of efficiency, fault tolerance, data distribution, and heterogeneity, providing ready access to large data sets.

Empirical testing on financial data validated the substantial advantages of our approach: directly loading in economic data, we achieved more than 5x compression compared to Parquet and ORC, and gained 6x speedup for queries running on Amazon Athena from raw data.

Moreover, with the integration to Apache Airflow, we achieved high stability, transparency, and ticketing of executions, which we needed to be able to use in production.

It is important to emphasize that the data was preserved in accordance with its original structure and semantics during the transformation. Integrity control, value loss prevention, time stamp preservation, and numeric field accuracy were all checked. This method achieves excellent storage efficiency while preserving analytical reliability by guaranteeing that the data is still appropriate for precise analytical processing even after extensive compression.

The flexibility and scalability of the proposed pipeline significantly expand its application capabilities. In addition to fundamental transformation and compression, the system can be extended for data profiling, anomaly detection, feature generation for machine learning models, or real-time streaming data processing. Building on open source software and cloud means the solution can evolve with infrastructure needs and the latest thinking in data engineering. Finally, the solution implementation is reproducible, productive, and cost-effective for businesses looking to modernize their data transformation practices. Future work will focus on supporting streaming processing, schema versioning, anomaly detection tools, and a real-time monitoring dashboard to make the system more convenient in a complex and dynamic data ecosystem.

AUTHOR CONTRIBUTIONS

Ye.K. – methodology, software, validation, investigation, writing-original draft preparation. I.M. – conceptualization, supervision writing-review.

COMPETING INTERESTS

The authors declare no competing interests.

REFERENCES

[1] Apache Software Foundation, *Apache Parquet Documentation*, 2023. [Online]. Available: https://parquet.apache.org/

[2] D. J. Abadi, P. A. Boncz, and S. Harizopoulos, “Column-Oriented Database Systems,” *Proc. VLDB Endow.*, vol. 2, no. 2, pp. 1664–1665, Aug. 2009, doi: 10.14778/1687553.1687609.

[3] Apache Software Foundation, “Apache Airflow Documentation,” 2024. [Online]. Available: https://airflow.apache.org/docs/

[4] Amazon Web Services, “Amazon EMR Developer Guide,” 2023. [Online]. Available: https://docs.aws.amazon.com/emr/

[5] Amazon Web Services, “Storage Best Practices for Data & Analytics,” 2022. [Online]. Available: https://docs.aws.amazon.com/whitepapers/latest/building-data-lakes/

[6] Amazon Web Services, “AWS Glue Documentation,” 2023. [Online]. Available: https://docs.aws.amazon.com/glue/

[7] Amazon Web Services, “Amazon Athena Documentation,” 2023. [Online]. Available: https://docs.aws.amazon.com/athena/

[8] M. Kleppmann, *Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems*, 1st ed. Sebastopol, CA: O’Reilly Media, 2017.

[9] M. Armbrust *et al.*, “Spark SQL: Relational Data Processing in Spark,” in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2015, pp. 1383–1394, doi: 10.1145/2723372.2742797.

[10] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, “Spark: Cluster Computing with Working Sets,” in *Proc. 2nd USENIX Conf. Hot Topics in Cloud Computing (HotCloud)*, 2010.

[11] Y. Mercadier, “Distance Measures for Probability Distributions,” 2022. [Online]. Available: https://distancia.readthedocs.io

[12] U. Kiran and J. Murphy, *Building Production Pipelines with Apache Airflow*. Birmingham, UK: Packt Publishing, 2020.

[13] M. Moazeni, “Automating Stock Market Data Pipeline with Apache Airflow, Spark, Postgres,” *Medium*, 2023. [Online]. Available: https://medium.com/@mehran1414/automating-stock-market-data-pipeline-with-apache-airflow-minio-spark-and-postgres-b67f7379566a

[14] A. Lakshman and P. Malik, “Cassandra: A Decentralized Structured Storage System,” *ACM SIGOPS Oper. Syst. Rev.*, vol. 44, no. 2, pp. 35–40, Apr. 2010.

[15] AWS Big Data Blog, “Best Practices for Using Amazon Athena,” Amazon Web Services, 2020. [Online]. Available: https://aws.amazon.com/blogs/big-data/best-practices-for-using-amazon-athena/

[16] Cloud Native Computing Foundation, “CNCF Cloud Native Landscape,” 2022. [Online]. Available: https://landscape.cncf.io/

[17] T. Chen and C. Guestrin, “XGBoost: A Scalable Tree Boosting System,” in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discov. Data Min.*, 2016, doi: 10.1145/2939672.2939785.

Yevhen Kyrychenko

Kyrychenko Yevhen is currently a Ph.D. student at the Department of Software Engineering, Yuriy Fedkovych Chernivtsi National University, Ukraine. He received his B.Sc. and M.Sc. degrees in Computer Science from Ivan Franko National University of Lviv. His research interests include cloud computing, big data technologies, and distributed systems.

ORCID ID: 0009-0005-6150-5410

Igor Malyk

Doctor of Physical and Mathematical Sciences, Professor, Head of Department of Mathematical Problems of Control and Cybernetics, Yuriy Fedkovych Chernivtsi National University, Chernivtsi, Ukraine.
Field of scientific interests: stochastic analysis, financial mathematics, machine learning, simulation of random processes.

ORCID ID: 0000-0002-1291-9167

Інформаційна технологія для стиснення та перетворення даних за допомогою Amazon EMR

Євген Кириченко[1,*], Ігор Малик[2]

[1]Кафедра програмного забезпечення комп'ютерних систем, Навчально-науковий інститут фізико-технічних та комп'ютерних наук, Чернівецький національний університет імені Юрія Федьковича, Чернівці, Україна

[2]Кафедра математичних проблем управління і кібернетики, Навчально-науковий інститут фізико-технічних та комп'ютерних наук, Чернівецький національний університет імені Юрія Федьковича, Чернівці, Україна

*Автор-кореспондент (Електронна адреса: kyrychenko.yevhen@chnu.edu.ua)

АНОТАЦІЯ Із зростанням обсягів обробки даних у різних сферах зростає й попит на застосунки, здатні ефективно управляти, опрацьовувати та трансформувати великі масиви інформації. Сучасні підходи до зберігання та обробки великих обсягів даних здебільшого базуються на універсальних текстових форматах, таких як CSV та JSON. Їхня популярність пояснюється простотою інтеграції та сумісністю з широким спектром програмних засобів. Проте під час роботи з великими наборами даних ці формати демонструють низьку ефективність, особливо при виконанні аналітичних запитів або масштабуванні систем. Відсутність вбудованої компресії, рядкова структура та нестача метаданих призводять до значних витрат часу й обчислювальних ресурсів, що створює суперечність між вимогами до швидкості й економічності обробки та технічними можливостями традиційних текстових форматів. Альтернативою виступають колонкові формати зберігання, такі як Parquet та ORC, які використовують компактну структуру, оптимізовану для швидких аналітичних запитів у розподілених обчислювальних середовищах. Завдяки вбудованим механізмам стиснення, ефективному кодуванню та індексації вони забезпечують значне зменшення обсягів даних і прискорюють обробку. Метою цього дослідження є розробка та експериментальна перевірка технології автоматизованого перетворення даних із неефективних текстових форматів у формати Parquet та ORC із використанням Apache Airflow та Amazon EMR. Запропонована архітектура передбачає створення хмарного пайплайна, що виконує конверсію даних і подальше збереження у форматах, орієнтованих на аналітичні навантаження. Система реалізована з використанням Apache Airflow для оркестрації процесів, Amazon EMR та Apache Spark для розподіленої обробки, AWS S3 як масштабованого сховища, AWS Glue для управління метаданими та Amazon Athena для SQL-доступу до перетворених даних. Такий підхід вирішує проблеми продуктивності, пропонуючи гнучке, надійне та економічно ефективне рішення, здатне адаптуватися до різних робочих сценаріїв і навантажень.

КЛЮЧОВІ СЛОВА інформаційна технологія, AWS, великі дані, розподілена обробка, стиснення даних.