Deep Science Publishing
https://doi.org/10.70593/978-93-49910-08-9_6

DeepScience
Open Access Books

Chapter 6: Architecting advanced data pipelines using real-time streaming and batch processing technologies

6.1. Introduction

Today, data is one of the most important company resources after human resources. It is therefore important to handle data in an adequate way. More and more organizations choose to advance their data strategies beyond simple data warehousing, so they can analyze bigger and bigger data that is produced in a more real-time basis, in a distributed manner, and with increasing variety. Over the past decade, different technologies have emerged which promise to be the answer to the big and fast data challenge. Organizations utilize these technologies in different ways, as well as different combinations (Dean & Ghemawat, 2008; Chauhan & Saxena, 2022; Arora & Talwar, 2023).

Traditional techniques that move data to a Data Warehouse to transform it with a set of users' business rules and create a DW model for easy business reporting have proven not only to be slow, but also not to have the capabilities needed to support advanced data analysis. Organizations have been attempting to augment their DW or replace it altogether with best practices that take advantage of flexible techniques. In doing this, they have learned that while the front-end report access process is a critical part of a data architecture, it is not the only component. In addition to the DW (often augmented with big data capabilities), organizations have created separate and unique big data environments to support custom application development, business intelligence, and advanced analytics. Data from both environments are often served to business reports, for true enterprise reporting. Both types of environments have different requirements and play different roles, and we believe that the solutions to the new data challenges require a new data architecture that takes advantage of the strength of both types of environments in combination, each with different technologies tailored to their respective strengths and weaknesses (Elgendy & Elragal, 2021; Malik, 2023).

Modern data pipelines can be classified in two main categories, according to how they process the data: **Batch Processing Pipelines** and **Stream Processing Pipelines**. Pipelines working on quanta of data that have been previously staged and moved in bulk into the processing system using batch processing technologies represent the traditional and most common data pipeline example where the data is made available in large amounts at predictable intervals. Upon consuming the new batch, a data processing job queries the data and executes various transformations that prepare it for querying and analysis .

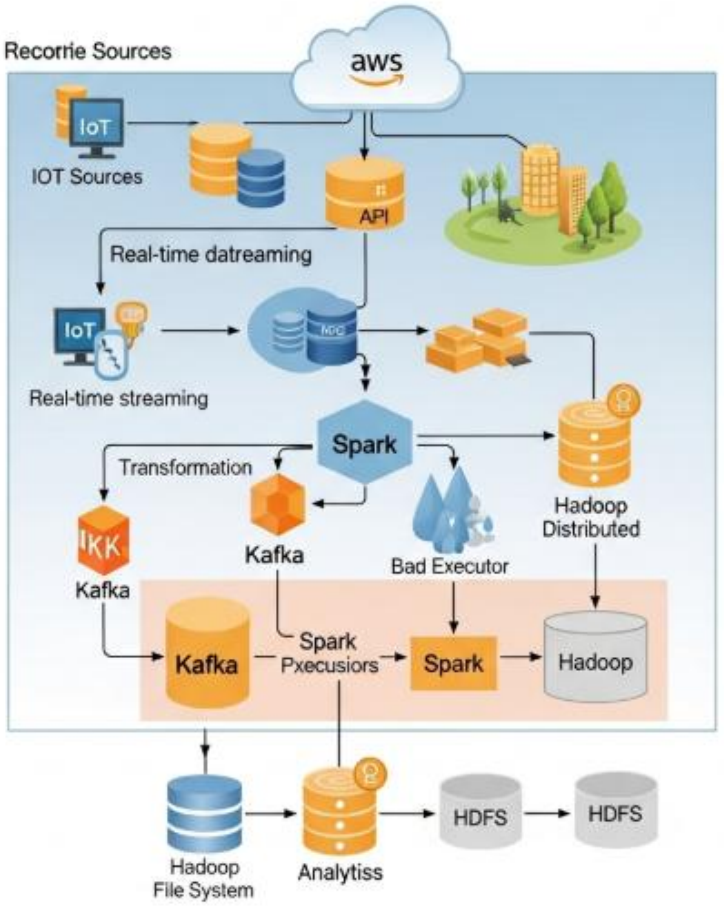

Fig 6.1: Advanced Data Pipelines

6.1.1. Background and Significance

Architecting data pipelines with real-time streaming and batch processing technologies has become a popular big data analytic use case in information technology organizations and led to the emergence of several new real-time data stream processing systems and frameworks. This also led to the emergence of numerous factory cloud data pipeline services both to build customer-facing products and monetization models using machine

learning. Business leadership and decision makers want to take a strategic approach to architect and evaluate their real-time data processing solutions with other traditional batch-based solutions.

Architecting data pipelines with hybrid data processing capabilities using batch and real-time stream processing technologies is one of the most requested capabilities from end customers and stakeholders coming from multiple industry verticals. What is interesting to notice is the new innovative technology solutions that came from implementation of such systems. Streaming and batch processing frameworks. Explainable ML Solutions. Interactive Business Intelligence Tools and Conceptual and Realizable System Architectures. In addition to opened solution assets, cloud platform vendors also organize cloud-based pipeline and streaming services to help industries evaluate the building of custom solutions and monetization offerings.

6.2. Understanding Data Pipelines

A **data pipeline** is a set of data processing elements that moves data from one system to another. Data pipelines present a flexible architecture to transform raw data from heterogeneous sources into information ready for analysis and to move it to presentation systems in an efficient way. A simple data pipeline can be made of a single processing step. However, most times the data modeling is substantial and it becomes important to break the process down into multiple processing steps, ensuring each individual step does one thing well. The pipeline will then take on a topology of connected processing nodes. Each step consumes a data stream from upstream and produces another data stream that is consumed by the downstream, thus providing the chaining of processing stages that characterize a pipeline.

6.2.1. Definition and Importance

Despite their growing prominence and application, data pipelines are not often clearly or academically defined, nor is their importance elaborated upon. A concise definition for data pipelines describes them as automated workflows that ingest and move data between storage and processing systems. It continues by explaining how they ingest data from one place, transform it if needed, and move it to another. That transformation process inside the pipeline is what differentiates the simple ETL and ELT procedures from a true data pipeline. In the case of a data pipeline, the automated process is what allows organizations to have real-time access to their data, while other patterns may only be able to apply batch processing, leaving data unreconciled for longer periods of time.

A data engineer elaborates further on why organizations depend on data pipelines for their operations. He states that, first and foremost, companies need pipelines to be successful. Without them, people would not be able to find the actionable insights necessary to create systematic changes to clients' businesses. He continues by explaining how pipelines are essential for any business solution that has any sort of real-time insight, whether that insight is being delivered after some sort of manual analysis or automated via a model that is sending out predictions, allowing machine learning processes to be automated and standardized. In short, data pipelines play an increasingly vital role in organizations for optimizing decision-making and communications, without which clients and management do not have the most accurate or thorough up-to-date information available to them.

6.2.2. Components of Data Pipelines

When designing data pipelines, there are several critical components to consider, depending on whether you are building batch or streaming nodes of the pipeline. The components we describe in the following sections are not unique to data pipelines and apply to many distributed systems. A data pipeline itself is a distributed system with some specific features that make it special. In the sections to follow, we take a deeper dive into some of the specific components of data pipelines. As we compare and contrast batch and streaming components, keep in mind that one major feature of such distributed systems that we are interested in at the level of data ingestion nodes is data volume, velocity, and variety.

The first component is the source system or the system from which data is being ingested into the data pipeline. Indeed, in data engineering projects involving enterprise applications, source systems usually expose their data through or they may have to write to location services, where they may directly write data in a file system, database, object store, and message queues. In the case of mobile applications, the consumers of data usually write to a centralized data store in the cloud or they may send event messages to a central messaging service. Sources are critical to a data engineering project because they enable us to ingest data from different systems and consolidate them into a common location for processing or analysis. They can have different formats like or different APIs such as or.

6.3. Real-Time Streaming Technologies

The highest available assurance of reducing the time delay between data creation and data availability is to use a real-time streaming architecture, which provides continuous queries over live streaming data from sources such as sensors, stock markets, social

media, or other events. In the traditional data processing batch processing world, data is usually accumulated in a store for some time interval and then a process is invoked to analyze the data, usually resulting in delayed data availability. But the nature of some applications requires that the catalog or store continuously reflect the current result, i.e., be continuously updated with the results. Lately, we have seen a huge focus on real-time streaming architectures due to our ability to create and collect data continuously.

The data collected is either continuously processed and updated in stores or continuously queried. A growing host of big data stores allow for continuous querying but there is a limitation to what can be achieved in the store. To counterbalance the limitations, the companies are also implementing solutions in the query languages or supporting constructs for the database. Most of these streaming computing systems have traditionally lacked a persistent mechanism to store data, even for quick lookups. As demand for low-latency data analysis has surged, so too has the number of products that are providing complex event processing, streaming technology.

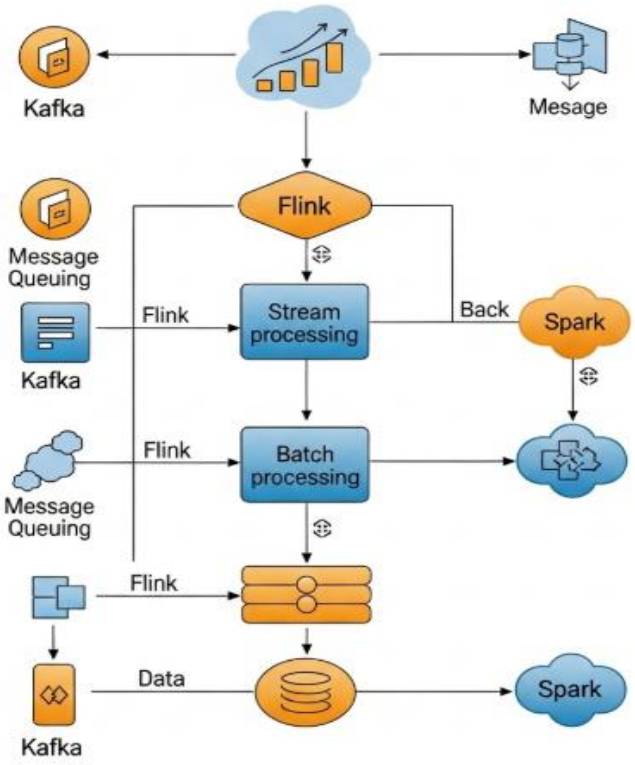

Fig 6.2: Real-Time Streaming Technologies

6.3.1. Overview of Streaming Technologies

Streaming technologies are powerful tools that have revolutionized the way we think about data engineering and the construction of real-time data-driven applications. Traditionally, event-driven data processing was a rare phenomenon, mostly relegated to extremely specialized or mission-critical applications. For the overwhelming majority of organizations, most data was collected into large aggregations, broadcast on a periodic basis, and processed through conventional ETL techniques. In fact, for most organizations, the key words in data management up until a few years ago were “batch” and “periodic.” The driving model for data access was “set at rest” rather than the more recent and more appropriate “continuous.” An explosion in the volume of data collected from numerous sensors, tweets, tracked user clicks, and transactions has created a need for applications that demand both immediate actions based on these data streams and/or the aggregation of the events over various temporal windows and time periods. Over the last few years, streaming technologies have evolved at a rapid tempo, driven by substantial contributions from cloud-based vendors, open-source projects, and projects sponsored by traditional data warehouse vendors. Streaming applications take many forms, from massively scaled event generation with high availability and fault tolerance to real-time data analysis on multiple large-scale data streams to real-time large-scale triggered event generation from numerous data streams.

6.3.2. Apache Kafka

Apache Kafka is arguably the most popular enterprise data streaming ecosystem, originally developed and later open-sourced. As a very high-throughput distributed commit log with partitioned publish/subscribe semantics, Kafka allows you to build many streaming-based applications in a fast, scalable, and fault-tolerant way. The producer is the process that writes events into topics. A producer can send events to the Kafka cluster using either a synchronous or an asynchronous API. If the producer uses the synchronous API, it blocks until a response is received from the Kafka cluster, and event records are published to the topic’s partition. If it uses the asynchronous API, events are sent to the broker without waiting for a response, which is recommended for better throughput performance. The consumer is the application that reads records from Kafka topics. Depending on the consumer type, it can use either a simple or a long polling APIs to read records.

Topics are split into partitions to achieve high throughput and scalability. When a producer sends a record to Kafka, it specifies only the topic to which the record belongs. Kafka then appends records to a partition based on a partitioning key of the message or by round-robin mapping records in the topic. On a partition, records are ordered. Each record has a unique offset/timestamp assigned by the Kafka broker when the record is

created, which allows clients to efficiently locate and fetch records within the partition. Partitions allow the Kafka cluster to distribute the recording load across many brokers to achieve high write/read throughput. By adding more partitions for a topic, you can easily scale its figures of merit. This design makes Kafka a 0-Latency Messaging system, enabling both the publishing of large volumes of data and processing with the lowest possible delay.

6.4. Batch Processing Technologies

Batch processing refers to executing a series of non-interactive tasks (or jobs) in a group while producing a batch of output data. In batch processing, data is collected over a given period and used as input to programs (also known as job or tasks) that process large volumes of data. Batch processing is advantageous for performing complex processing over huge volumes of data. It is also useful when it takes a long time to generate output data. As an example, batch processing generates the output of advancing the interest on a savings account. The interest is calculated once a week based on the balance present at the end of each day and is usually small in value compared to the total balance. It typically takes a couple of hours to run the batch program and generate the interest report for hundreds of thousands of bank accounts. Batch Processing is useful when processing large volumes of structured data, and thus it is typically used in Data Warehousing, ETL, Reporting, and Business Analytics.

Apache Hadoop is an open-source batch processing framework that runs on distributed computer clusters. It utilizes and expands on the MapReduce paradigm for creating large-scale distributed Data Processing jobs. Hadoop was designed to work with commodity hardware. Instead of having a high-availability solution, where every node is a high-end server, it works on a cluster of cheap servers. The data in Hadoop is replicated across the nodes to provide fault tolerance and higher reliability. Hadoop is primarily used for hosting large-scale ETL, Data Processing Pipelines, and Data Warehouse workloads. Hadoop is not a good fit for Interactive Queries, which are better run on OLAP Services. Hadoop excels at scheduled Batch Jobs and at volumes where traditional ETL tools fail.

6.4.1. Overview of Batch Processing

Batch data processing refers to the jobs that act on a stream of data that is finite, which is received over time at any random actual time but is typically not available in real time, while remaining inert for possibly a long time before being analyzed to help in the actual decision making. The batch job is typically submitted and scheduled to run for a predetermined duration of execution, until completion, possibly on a dedicated

computing cluster. Batch processing in data analytics is the original way in which data analytics took place, which enabled the democratization of analytics. Due to the discrete workloads, batch processing is typically more economical in terms of computing costs and can free up resources during idle time, provided there are other workloads available that can afford to wait for processing. Additionally, it allows batch data processing technologies to compete with other technologies like traditional business intelligence tools in terms of the time taken to produce insights.

Batch processing is also responsible for the reduced costs of data storage incurred by organizations that choose to store data for long periods of time for regulatory compliance by moving such data into lower cost cloud storage, and to have it processed there using batch processing technologies, which traditionally charged customers not on the typical metered usage of real time resources but rather on the storage utilization during the company's exact peak storage duration, if any during the month. Although near real time processing of data is possible with the technology with results issued in seconds, even milliseconds, after data is newly ingested, this would obviously incur much higher costs by virtue of reducing system parallelism.

6.4.2. Apache Hadoop

In the year 2005, a number of research and technical papers introduced systems such as MapReduce for computation and BigTable for data storage, which could be scaled to thousands of nodes, to process huge datasets. These papers opened the floodgates for the massive amounts of research being done today in the general purpose parallel/distributed processing systems space, and thousands of systems have been created. At the end of the day, however, there can be only one or a few systems that will be widely used in the industry, regardless of how many others are created. In this domain, it appears that the open-source project has emerged as the clear leader at the current time. It is a combination of the distributed file system support as well as the MapReduce implementation. Based largely on the earlier papers, it was created in 2005 by individuals.

It has continued to advance significantly in recent years. Some of the more significant advances in the system have been the addition of support for streaming data processing with the introduction of components such as Streaming and Flume, the addition of a data warehousing solution called Hive, in addition to other projects including Pig and Jaql, and the evolution of the systems as specializations of similar systems. There is even an implementation of MapReduce for the Windows environment! Supporting technologies are now widely deployed and used in production applications, and it is being used successfully in a variety of scenarios in addition to search engine indexing such as for social network analysis and malware detection.

6.5. Architectural Patterns for Data Pipelines

Architecting a data pipeline is more about solving a technical problem in the right way rather than employing a particular technology or deployment model. It is, therefore, quite useful for both architects and developers to understand the common architectural patterns for data pipelines across industry verticals. These architectural patterns broadly fall within the umbrella of the lambda and kappa architectures. Even though the lambda architecture is an older paradigm, it is a very popular model for data pipelines that need to serve both real-time and historical analytical use cases – especially for companies in the online retail, advertising, and entertainment business. The kappa architecture is a relatively newer paradigm that has evolved out of the limitations and constraints of the lambda architecture. Even though the kappa architecture is a popular model for operational applications that need real-time analytics, it is often seen as a very niche model compared to the lambda architecture. The lambda architecture has also evolved over the years into several deployment patterns that combine concepts from both architectures.

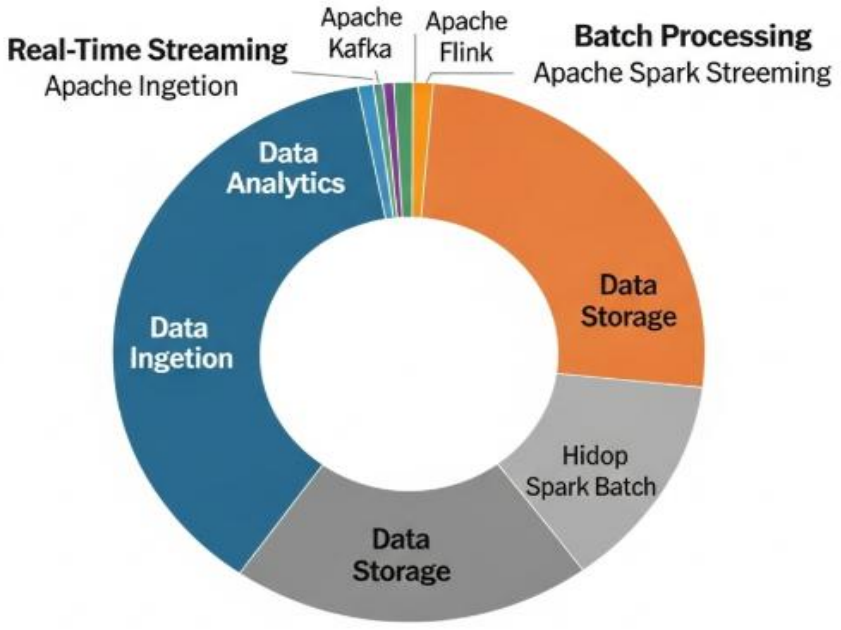

Fig : Advanced Data Pipelines Using Real-Time Streaming and Batch Processing Technologies

Lambda architecture defines the concept of batch and speed layers. The batch layer not only manages the master data store that is used to generate results for queries over the historical data but also runs at regular intervals very large batch jobs on the historical data to generate the results which will be used most frequently by the serving layer to respond to queries. The speed layer serves the most recent data while the month or year

old results are served from the batch layer. The serving layer combines the results from both the batch and speed layers to respond to queries, thus acting as the glue that integrates both the batch and speed layers. Hence, the lambda architecture integrates both batch and real-time processing models into a single analytical layer.

6.5.1. Lambda Architecture

Distributed systems have been indispensable for data processing, where we run various workloads and handle different types of data across many servers to speed up the processes. With that, we have big storage systems, distributed clusters, communication abstractions, and frameworks built for batched data. Although storing the information of the measured phenomena is strongly necessary, having a pure batch processing system cannot satisfy the low latency response requests required when dealing with Data-in-motion.

The Lambda Architecture, proposed at the beginning of the Big Data era, is the first architecture that ties together and generalizes the entire heap of new technologies in order to process data for near real time and for historical analysis, all together in a simple way. This architectural pattern reconciling the additive data evolution pattern proposes to process data in both real time as well as using the good-old batch paradigm; everything to manage to answer to those queries that are put on top of the stored data.

That being said, the typical architecture relies on three layers: Firstly, the data is ingested into the system from the sources using an ingestion layer – like an ETL, often using a Stream Processing system. This huge data flow is then split and the data events are stored in two different systems: a realtime (or speed) layer with Raw Non-aggregated data, and a batch layer with the Aggregated results over a time interval for the analysis. The batch system will never be enabled to respond to near real time queries. The last component of the Lambda Architecture is the Serving Layer that manages to deliver responses to the user queries, having them equipped with both the real time and batch pre-computed data.

6.5.2. Kappa Architecture

In data intensive applications, events are frequently ingested at high volume and high velocity. In many cases, a record of old events is retained for a (short) period of time so they can be re-consumed if errors are found or business logic changes. Kappa Architecture recognizes this common set of practical concerns, and introduces the idea of a system in which all processing happens in a stream processing layer. The status of the long-term views is stored within a key-value distributed storage system. Accordingly

when a user request comes, it accesses the view in the KV store. Kappa Architecture attempts to simplify the Lambda Architecture approach in an effort to make building and operating data intensive applications easier. Specifically, it posits that the batch processing layer essentially boils down to, and is dramatically simplified by, the ability to recompute views of select key-value pairs when they become stale, and that embedded in the stream processing layer is a system for storing the resulting views, and for recalling them when necessary. The coalescing of these facets within the stream processing layer also greatly simplifies maintaining consistency across batch and streaming data, as the Kappa Architecture suggests that they are one and the same. In fact, both highlights and contributes to the capability of stream processing as an event by event computation model on data.

Kappa Architecture results in a simpler architecture, and it eliminates the batch layer, the complexity from Lambda Architecture. However, this is only plausible when the business cases suit. If a query requests information about the last five months for example, if the stream layer in the Kappa Architecture is the only one possible then, then it is impossible to answer this request without computing a batch first. It's a matter of computing using Kappa Architecture and then answering the request or doing nothing and then answering the request as querying a view stored.

6.6. Conclusion

This chapter highlighted several best practices and various technologies to implement advanced data pipelines. These pipelines can use both real-time streaming and batch processing technologies to implement ETL for analytical applications. Each of the technologies in the pipeline can be implemented on standardized platforms to provide the capability of pushing the workloads down to big data processing engines, which run on clusters of commodity machines.

Through the various sections of this chapter, we provided insights about choices to consider while architecting data pipelines. We applied these best practices while also reaping the benefits from the advanced capabilities of the various technologies that were discussed, to solve real analytic use cases at scale efficiently. We believe that with enterprise data moving to the cloud, there will be an increased demand for building advanced data pipelines, especially using serverless, managed services, as the required experience and expertise to run and maintain the individual technologies and their optimization for scale would be minimized. Such pipelines will also manage the complexity of serving the reliable analytics demands through a mix of traditional ETL, Lambda, and Kappa architectures.

As enterprise data processing starts taking advantage of the elasticity, cost-effectiveness, and sheer compute power offered by cloud infrastructure, we believe there will be a growth in not only the capacity and speed of the data pipelines being built, but also the volumes of diverse data flowing through them. Virtualization of the various advanced technologies, both on-premise as well as in the cloud, coupled with close integration between them, will help implement these data pipelines. Serverless computing will stimulate demand for real-time pipelines that will enable enterprises to seamlessly process enterprise data to meet the operational and analytical demands.

6.6.1. Future Trends

As organizations continue their digital transformation journey, the demand for advanced data analytic solutions is expected to skyrocket. In parallel, significant increases in the data volume as well as velocity are fueling the emergence of faster and faster high-quality data pipelines that are built using groups of diverse pipeline design patterns. More specifically, the design of faster enterprise pipelines must address the integration of diverse data sources that span real-time streams such as Internet of Things devices, external application programming interfaces, clicks, orders, stocks, social media, and mobile and web logs among many others; internal operational batch and near-real-time updates, and historical archives of data stored in cold batch repositories.

Moreover, this growing data pipeline complexity must be addressed with more flexible and faster integrated data pipeline development frameworks that are architected using the best-of-breed library of reusable high-quality design components and tools that simplify the pipeline design, create pipelines from a library of pipeline components that support any type of source and sink integration, and support diverse data transformations including cleaning, data preparation, enrichment, and model score and data publishing, especially for batch and stream hybrid design patterns. In parallel, we expect to see the proliferation of enterprise-grade data ops tools that support collaborative DevOps and MLOps teams, empowered by best-of-breed metadata services, library MLOps, data ops, source control, CI/CD, and self-service data provisioning tools.

Finally, although machine learning capabilities are slowly becoming easier to integrate into enterprise applications, they remain arduous, especially for batch and real-time hybrid pipelines that implement advanced business logic. To help reduce this complexity, we expect to see the growth in adoption of the library of reusable, documented, open source pre-trained ML components.

References

Dean, J., & Ghemawat, S. (2008). MapReduce: Simplified data processing on large clusters. Communications of the ACM, 51(1), 107–113.

Malik, A. (2023). Zero Trust in Intelligent Cloud Architectures. Journal of Cybersecurity & Cloud Technologies, 2(2), 60–73.

Chauhan, A., & Saxena, V. (2022). Serverless MLOps pipelines: Architectures and strategies. International Journal of Cloud Applications, 11(3), 149–168.

Elgendy, N., & Elragal, A. (2021). Big data analytics: A literature review paper. Journal of Computer Science and Technology, 31(3), 381–396.

Arora, A., & Talwar, A. (2023). Trustworthy AI systems for cloud-native environments. Proceedings of the ACM Conference on Fairness, Accountability, and Transparency (FAccT).