Available online www.ejaet.com

European Journal of Advances in Engineering and Technology, 2019, 6(11):64-71

Research Article ISSN: 2394 - 658X

Incremental Processing with Structured Streaming

Ravi Shankar Koppula

ravikoppula100@gmail.com

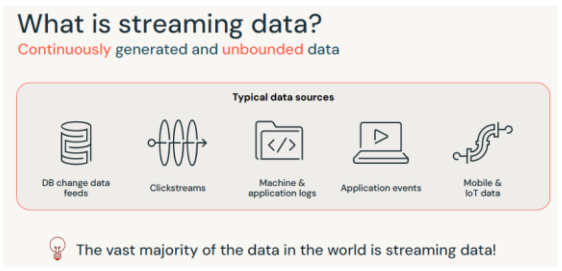
ABSTRACT

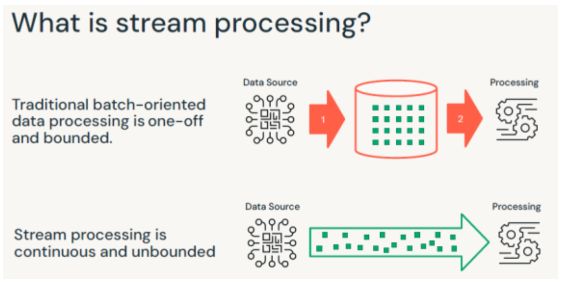
This paper presents a study on the utilization of Structured Streaming, which is a stream processing framework that uses the Apache Spark SQL engine. The study focuses on transitioning from traditional batch processing to stream processing in order to handle real-time datasets that have no size limitations. Structured Streaming provides a user-friendly, low-latency, and resilient approach to processing streaming data, addressing common issues like out-of-order data, managing states, and ensuring fault tolerance. It operates on a data model that treats data as a continuously expanding table and utilizes micro-batch and continuous execution modes to optimize throughput and latency. Furthermore, it introduces a consistent, fault-tolerant API that integrates batch and stream processing. The integration with Delta Lake enhances this framework by offering robust storage solutions, efficient handling of data discrepancies through features like schema evolution, and ensuring correctness through append-only operations and comprehensive support for stream-stream joins. The paper concludes that the combined usage of Structured Streaming and Delta Lake provides a powerful framework for analyzing and processing big data streams in real-time, accommodating various applications such as alerts, anomaly detection, and recommendation systems

Key words: Structured Streaming, Spark Streaming, Delta Lake Streaming, Stream Processing

INTRODUCTION

Streaming data refers to a type of data that is generated continuously and, without any limits. It flows in an uninterrupted manner. Its rapidly becoming the primary form of data worldwide. Examples of streaming data include updates on media, website traffic readings from sensors and financial transactions. Streaming data makes up the majority of the information we encounter on a basis. To effectively handle, process and gain insights from this vast amount of information specialized tools and techniques are required. Managing and analyzing streaming data efficiently will be crucial in utilizing its potential as our world becomes more interconnected and driven by data. In contrast, to batch-oriented data processing that handles information in one operation stream processing is an ongoing process.

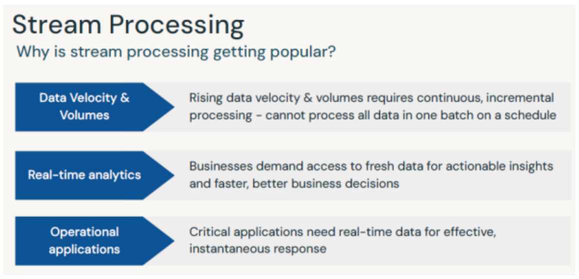

What is streaming data?
Continuously generated and unbounded data
Typical data sources
DB change data feeds
Clickstreams
Machine & application logs
Application events
Mobile & IoT data
The vast majority of the data in the world is streaming data!


What is stream processing?
Traditional batch-oriented data processing is one-off and bounded.
Data Source
1
2
Processing
Stream processing is continuous and unbounded
Data Source
Processing

The increasing trend of stream processing is attributed to its efficiency in managing vast volumes of live data, its capability of providing instantaneous analysis, and its versatility in various sectors.

- Data Velocity & Volumes - Rising data velocity & volumes requires continuous, incremental processing - cannot process all data in one batch on a schedule

- Real-time analytics - Businesses demand access to fresh data for actionable insights and faster, better business decisions
- Operational applications - Critical applications need real-time data for effective, instantaneous response [1]


Stream Processing

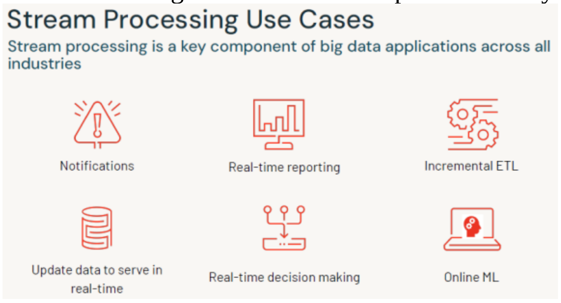
Why is stream processing getting popular?

Data Velocity & Volumes	Rising data velocity & volumes requires continuous, incremental processing - cannot process all data in one batch on a schedule
Real-time analytics	Businesses demand access to fresh data for actionable insights and faster, better business decisions
Operational applications	Critical applications need real-time data for effective, instantaneous response

Stream Processing Use Cases

Stream processing is a key component of big data applications across all industries

1. Real-time alerts and notifications are essential for various applications that require immediate action.
2. Stream processing enables the generation of timely notifications based on incoming data.
3. These notifications can be sent to users or other systems through various communication channels.
4. Email, SMS, and push notifications are commonly used methods for delivering real-time alerts.
5. Stream processing use cases often involve monitoring systems, fraud detection, and anomaly detection.
6. By continuously processing streaming data, organizations can quickly respond to critical events.
7. Stream processing can also be leveraged for predictive analytics and recommendation systems.
8. The ability to process data incrementally allows for more efficient and scalable processing.
9. Incremental processing with structured streaming enables real-time updates to analytics dashboards. [2]


Stream Processing Use Cases

Stream processing is a key component of big data applications across all industries

Overall, stream processing offers a wide range of use cases for handling real-time data. About important events or updates is crucial in our fast-paced world. It allows us to stay informed and take immediate actions when needed. Whether it is breaking news, important emails, or app notifications, having real-time notifications can greatly enhance our productivity and keep us connected to what matters most. By being promptly notified, we can efficiently manage our time and responsibilities, ensuring nothing important slips through the cracks. In this digital age, where information is constantly being shared and updated, real time notifications provide us with the necessary tools to stay on top of things and make well-informed decisions. So, let us embrace the power of real-time notifications and make the most out of our connected lives.

Real-time reporting has revolutionized the way information is communicated and analyzed in today's fast-paced world. With its ability to instantly collect data, process it, and generate comprehensive reports, real-time reporting has become an invaluable tool for businesses, organizations, and individuals alike. This technology allows for the immediate monitoring of various metrics, such as sales figures, website traffic, customer feedback, and social media engagement, providing invaluable insights and actionable intelligence. The speed and accuracy of real-time reporting enable individuals and decision-makers to make informed choices, adjust strategies, and seize opportunities as they arise. By harnessing real-time reporting, companies can stay ahead of the competition, anticipate trends, and adapt quickly to changing market conditions. In an increasingly data-driven world, real-time reporting is the key to staying agile, responsive, and successful.

Incremental Extract, Transform, and Load (ETL) is a data integration approach that focuses on processing and transferring data in small, incremental batches. It minimizes processing time and resources by only working with the changes or updates that have occurred since the last ETL process. This approach provides near-real-time data updates and improves scalability by only processing changes rather than the entire dataset. However, implementing incremental ETL requires careful design and can be more complex to maintain and troubleshoot. Despite these challenges, it offers numerous benefits and is increasingly popular for improving data processing efficiency, timeliness, and scalability for better decision-making and business outcomes.

Update the existing data to serve in real-time for online machine learning applications, ensuring quick and efficient processing of new information as it becomes available.

Real-time decision making is a crucial process in which individuals and organizations use up-to-date data and information to make strategic and tactical choices promptly. This dynamic approach allows for swift analysis, assessment, and implementation, enhancing efficiency and responsiveness in various industries and sectors. Real-time decision making empowers individuals to act swiftly and confidently, ensuring timely responses to changing circumstances and maximizing opportunities for success. By harnessing the power of real-time data and analytics, decision-makers can gain valuable insights, optimize resource allocation, mitigate risks, and seize competitive advantages. Invested with the ability to adapt and pivot swiftly, organizations can navigate complex and fast-paced environments, positioning themselves for continued growth and resilience in today's dynamic world.

Online machine learning is a subfield of machine learning that focuses on the development of algorithms and models that can learn from and make predictions on streaming data. Unlike traditional batch learning, where the entire dataset is used to train a model offline, online ML algorithms are designed to continuously update and adapt their models in real time as new data becomes available. This allows for more dynamic and responsive models that can quickly adapt to changing patterns and trends in the data. Online ML finds applications in various domains, such as recommendation systems, fraud detection, clickstream analysis, and anomaly detection. It has become increasingly popular in recent years due to the growing availability of streaming data and the need for real-time decision-making in many industries.

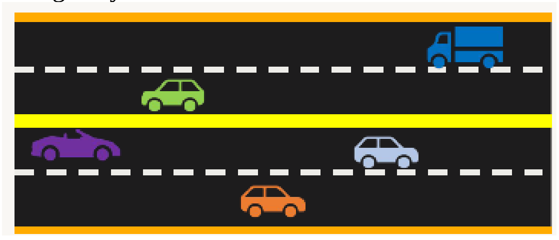
Bounded vs. Unbounded Dataset

Bounded Data:

- Has a finite and unchanging structure at the time of processing.
- The order is static.
- Analogy: Vehicles in a parking lot.

Unbounded Data:

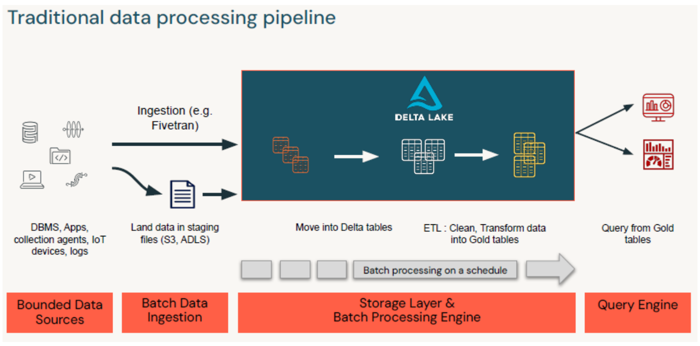
- Has an infinite and continuously changing structure at the time of processing.
- The order not always sequential.
- Analogy: Vehicles on a highway



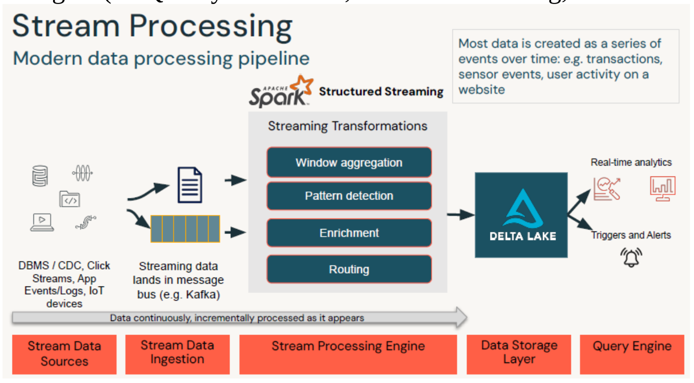
Batch vs. Stream Processing

Batch Processing:

Generally, refers to processing & analysis of bounded datasets (ie. size is well known, we can count the number of elements, etc.)
Typical of applications where there are loose data latency requirements (ie. day old, week old, month old).
This was traditional ETL from transactional systems into analytical systems


Traditional data processing pipeline
Ingestion (e.g. Fivetran)
DELTA LAKE
DBMS, Apps, collection agents, IoT devices, logs
Land data in staging files (S3, ADLS)
Move into Delta tables
ETL : Clean, Transform data into Gold tables
Query from Gold tables
Batch processing on a schedule
Bounded Data Sources
Batch Data Ingestion
Storage Layer & Batch Processing Engine
Query Engine

Stream Processing:
Datasets are continuous and unbounded (data is constantly arriving, and must be processed as long as there is new data)
Enables low-latency use cases (ie. real-time, or near real-time)
Provides fast, actionable insights (ie. Quality-of-Service, Device Monitoring, Recommendations, etc.) [3]


Stream Processing
Modern data processing pipeline
Most data is created as a series of events over time: e.g. transactions, sensor events, user activity on a website
Spark Structured Streaming
Streaming Transformations
Window aggregation
Pattern detection
Enrichment
Routing
DELTA LAKE
Real-time analytics
Triggers and Alerts
DBMS / CDC, Click Streams, App Events/Logs, IoT devices
Streaming data lands in message bus (e.g. Kafka)
Data continuously, incrementally processed as it appears
Stream Data Sources
Stream Data Ingestion
Stream Processing Engine
Data Storage Layer
Query Engine

SIMILARITIES AND DIFFERENCES BETWEEN STREAM AND BATCH PROCESSING:

Similarities:

- Both have data transformation
- The results of the streaming job are frequently accessed through batch jobs.
- Stream processing commonly incorporates batch processing, specifically micro-batching.

Differences:

- The batch processing engine handles large batches of data, while the stream processing engine processes data row by row or in mini-batches for bounded data sets.
- In the case of an unlimited dataset, batch processing is conducted through several iterations, while Stream processing is carried out in smaller batches.
- Computation queries are carried out once during batch processing and multiple times during stream processing.[4]

Advantages of Stream Processing

Why use streaming (vs. batch)?

- A more intuitive way of capturing and processing continuous and unbounded data
- Reducing latency for applications and use cases that require quick response times.
- Improved resilience by implementing a checkpointing mechanism
- Automated accounting of fresh information.
- Increased utilization of computing resources and the ability to scale up through ongoing and gradual data processing.

Challenges of Stream Processing

- Stream processing is not easy
- Processing out-of-order data based on application timestamps (also called event time)
- Maintaining large amounts of state
- Processing each event exactly once despite machine failures
- Handling load imbalance and stragglers
- Determining how to update output sinks as new
- events arrive
- Writing data transactionally to output systems

Structured Streaming

Structured Streaming is a scalable and fault-tolerant stream processing engine built on the Spark SQL engine. It provides high-level APIs in Scala, Java, and Python, thus making it effortlessly accessible and adaptable to a diverse community of developers worldwide. With its advanced features and reliable architecture, Structured Streaming enables seamless and efficient handling of real-time data streams, ensuring optimal performance and accuracy of data processing. Whether you are a seasoned developer or a beginner, Structured Streaming empowers you to effortlessly harness the potential of streaming data and unlock novel opportunities for data analysis and application development.

How Structured Streaming Works

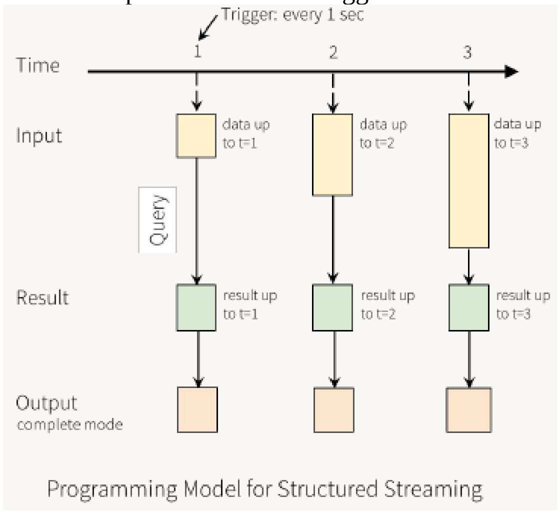
Incremental Updates:- Data stream as an unbounded table. Streaming data is usually coming in very fast, with a continuous flow of information. The remarkable and innovative concept behind Spark Structured Streaming lies in processing infinite data seamlessly, treating it as incremental table updates that are constantly evolving and expanding with each new piece of data that arrives in the stream. This groundbreaking approach enables the real-time analysis and transformation of data, empowering organizations to harness the full potential of their streaming data sources for actionable insights and improved decision-making. With Spark Structured Streaming, the possibilities are boundless, as it unlocks the ability to perform complex analytics, aggregations, and computations on never-ending data streams, providing an unparalleled level of flexibility and adaptability to handle the dynamic and ever-changing nature of streaming data. By seamlessly capturing, processing, and updating data in an incremental and efficient manner, Spark Structured Streaming paves the way for real-time applications, machine learning models, and diverse use cases that rely on up-to-the-second insights and accurate predictions. Whether it's monitoring live events, detecting anomalies in real-time, or continuously updating a live dashboard with the most recent data points, Spark Structured Streaming is the cornerstone of modern data processing, enabling the seamless integration of real-time streaming data into a structured and manageable format. Embracing the power of incremental updates, Spark Structured Streaming ensures that organizations can stay ahead in this era of overflowing data, unlocking the true value and potential of unbounded data streams.[5]

Micro-batch Execution: Accumulate small batches of data and process each batch in parallel.

Continuous Execution: Continuously listen for new data and process them individually.

Execution mode:

1. An input table is defined by configuring a streaming read against source.
2. A query is defined against the input table.
3. This logical query on the input table generates the results table.
4. The output of a streaming pipeline will persist updates to the results table by writing to an external sink.
5. New rows are appended to the input table for each trigger interval.

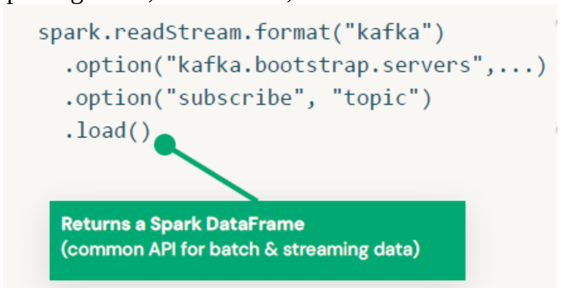


Anatomy of a Streaming Query

A streaming query is composed of various components and stages. Each component and stage play a crucial role in the incremental processing of structured streaming. In this section, we will delve into the details of each component and stage to understand their significance in the context of incremental processing with structured streaming.

Core concepts:

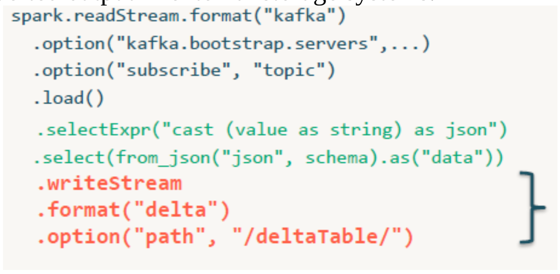
Input sources: Specify where to read data from. Spark supports Kafka and file sources. Databricks runtimes include connector libraries supporting Delta, Event Hubs, and Kinesis



```
spark.readStream.format("kafka")
  .option("kafka.bootstrap.servers",...)
  .option("subscribe", "topic")
  .load()
```

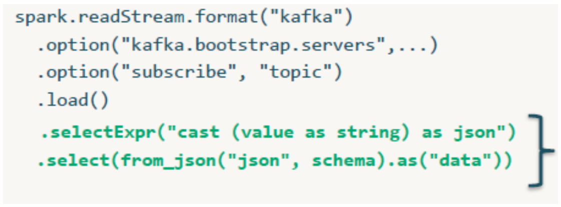
Returns a Spark DataFrame
(common API for batch & streaming data)

- **Sinks:** Store the converted output in external storage systems.



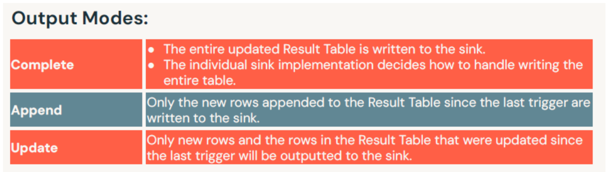
```
spark.readStream.format("kafka")
  .option("kafka.bootstrap.servers",...)
  .option("subscribe", "topic")
  .load()
  .selectExpr("cast (value as string) as json")
  .select(from_json("json", schema).as("data"))
  .writeStream
  .format("delta")
  .option("path", "/deltaTable/")
```

- **Transformations & actions:** 100s of built-in, optimized SQL functions for efficient data processing in structured streaming.like from_json. In this example, cast bytes from Kafka records to a string, parse it as JSON, and generate nested columns



```
spark.readStream.format("kafka")
  .option("kafka.bootstrap.servers",...)
  .option("subscribe", "topic")
  .load()
  .selectExpr("cast (value as string) as json")
  .select(from_json("json", schema).as("data"))
```

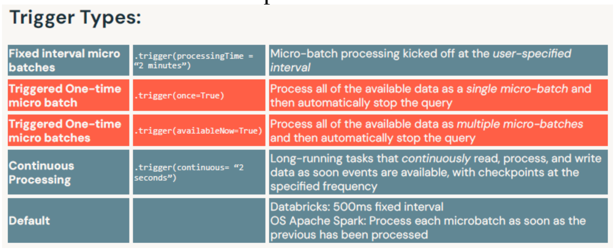
- **Checkpoint location:** For tracking the progress and state of a streaming query. progress of the query
- **Output Mode:** Defines how the data is written to the sink; Equivalent to "save" mode on static DataFrames



Output Modes:

Complete	• The entire updated Result Table is written to the sink. • The individual sink implementation decides how to handle writing the entire table.
Append	Only the new rows appended to the Result Table since the last trigger are written to the sink.
Update	Only new rows and the rows in the Result Table that were updated since the last trigger will be outputted to the sink.

- **Trigger:** Defines how frequently the input table is checked for new data; • The data is checked for new records at regular intervals; The trigger interval can be adjusted to optimize performance. Each time a trigger fires, Sparks check for new data and updates the results



Trigger Types:

Fixed interval micro batches	`.trigger(processingTime = "2 minutes")`	Micro-batch processing kicked off at the *user-specified interval*
Triggered One-time micro batch	`.trigger(once=True)`	Process all of the available data as a *single micro-batch* and then automatically stop the query
Triggered One-time micro batches	`.trigger(availableNow=True)`	Process all of the available data as *multiple micro-batches* and then automatically stop the query
Continuous Processing	`.trigger(continuous= "2 seconds")`	Long-running tasks that *continuously* read, process, and write data as soon events are available, with checkpoints at the specified frequency
Default		Databricks: 500ms fixed interval OS Apache Spark: Process each microbatch as soon as the previous has been processed

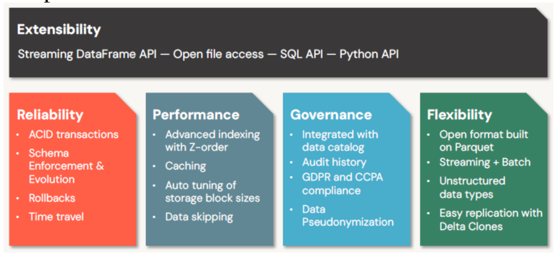
Benefits of Structured Streaming

Unification: The Unified API for Batch and Stream Processing allows for the use of the same API for both batch and stream processing. This API supports various languages including Python, SQL, and Spark's other supported languages. It also enables the utilization of Spark's built-in libraries, including machine learning libraries, in a streaming context. This capability provides developers with a wide range of options for data manipulation and analysis within the streaming query. Most operations on a streaming DataFrame are identical

to a static DataFrame, except for the output mode and the option to specify a watermark column DataFrame. There are some exceptions to this, for example, sorting is not supported with streaming data.
End-to-end fault tolerance: Structured Streaming ensures end-to-end exactly-once fault-tolerance guarantees through checkpointing. In case of failures; the streaming engine attempts to restart and/or recover from the last saved checkpoint and reprocess the data. This approach requires Replayable streaming source such as cloud-based object storage and pub/sub services. In addition to that, the system employs redundant fault recovery mechanisms, including automatic error detection and correction algorithms. This ensures a seamless and uninterrupted streaming experience, even during unpredictable events. Moreover, a comprehensive monitoring system constantly evaluates the health and performance of the streaming pipeline, providing real-time insights and alerts to the operators. With these advanced features and measures in place, the system further enhances its ability to handle unexpected scenarios and maintain the integrity of the data. As a result, Structured Streaming not only guarantees fault tolerance but also enables efficient and reliable data processing. Lastly, the utilization of idempotent sinks ensures that multiple writes of the same data, identified by the offset, do not result in duplicates being written to the sink. This additional layer of redundancy further enhances the stability and accuracy of the streaming process.
Managing Data Discrepancies: Providing assistance for handling data that arrives out of sequence and enabling aggregation queries based on event time windows and watermarking, which empowers users to establish a threshold for handling delayed data.[6]

STRUCTURED STREAMING WITH DELTA LAKE

Delta Lake Benefits: An open format storage layer built specifically for the innovative and powerful lakehouse architecture. Offering a wide range of advantages and functionalities, Delta Lake ensures seamless operations with its support for the highly efficient streaming dataframe API. Additionally, it enables open file access, granting users the ability to effortlessly interact with their data. With the inclusion of the versatile SQL API, executing complex queries becomes a breeze, while the Python API provides flexibility and convenience for data manipulation. Through its comprehensive feature set, Delta Lake truly revolutionizes data management and empowers users to unlock the full potential of their lakehouse architecture.


Extensibility
Streaming DataFrame API — Open file access — SQL API — Python API
Reliability
ACID transactions
Schema Enforcement & Evolution
Rollbacks
Time travel
Performance
Advanced indexing with Z-order
Caching
Auto tuning of storage block sizes
Data skipping
Governance
Integrated with data catalog
Audit history
GDPR and CCPA compliance
Data Pseudonymization
Flexibility
Open format built on Parquet
Streaming + Batch
Unstructured data types
Easy replication with Delta Clones

Streaming from Delta Lake: When Using a Delta table as a streaming source, Each committed version represents new data to stream. Delta Lake transactions logs identify the version’s new data files. Structured Streaming assumes append-only sources. Any non-append changes to a Delta table causes queries streaming from that table to throw exceptions. Set delta.appendOnly = true to prevent non-append modifications to a table. Use Delta Lake change data feed to propagate arbitrary change events to downstream consumers. In addition, Delta Lake provides built-in support for stream-stream joins, enabling users to merge multiple streams on the fly. This functionality allows for real-time data integration and aggregation, providing a comprehensive solution for streaming data processing. Moreover, Delta Lake's schema evolution capabilities ensure that data can be seamlessly added or modified over time, without sacrificing compatibility with existing data structures. With its robust features and seamless integration with popular streaming frameworks, Delta Lake is a powerful tool for managing and processing streaming data.
You have the ability to control and manage the rate at which micro-batches are processed by adjusting the maximum number of records or the maximum data size. By specifying these options in the DataStreamReader, you can effectively limit the input rate and ensure that it aligns with your desired parameters. This level of control allows for more efficient and optimized processing of micro-batches, resulting in enhanced performance and improved overall data management.
maxFilesPerTrigger: Maximum number of files that can be read per micro-batch. By default, this value is set to 1,000, but it can be adjusted according to your needs and the size of your dataset. Increasing this value can improve the processing speed, especially when dealing with large datasets. However, keep in mind that increasing it too much can put a strain on your system resources. So, it is important to find the right balance and consider the capabilities of your hardware while making this adjustment.

maxBytesPerTrigger: Soft limit to amount of data read per micro-batch. This parameter allows you to control the size of each micro-batch by setting a maximum limit on the amount of data that can be read in a single batch. This is particularly useful when dealing with large datasets, as it helps prevent memory issues and ensures a smooth and efficient processing of data. By specifying a value for maxBytesPerTrigger, you can optimize the performance of your data processing tasks and ensure that the micro-batches are sized appropriately for your specific use case.[7]

Delta Live Tables pipelines provide automatic tuning options for rate limiting, which means that it is recommended to refrain from explicitly setting these options for your pipelines. By leveraging the auto-tuning capabilities of Delta Live Tables, you can optimize the rate limiting process and ensure efficient pipeline operations.

Each micro-batch written to the Delta table is committed as a new version. Delta Lake supports both append and complete output modes. Append is most common. Complete replaces the entire table with each micro-batch. It can be used for streaming aggregations or when the entire result needs to be reprocessed. This output mode guarantees that each micro-batch's result will be stored as a complete snapshot of the table at that point in time. This ensures consistency and allows for easy rollbacks or rewinds in case of failures. streaming queries that perform arbitrary aggregations on streaming data.[8]

SUMMARY

The article explores the insufficiency of conventional batch processing when it comes to managing continuous and limitless data, emphasizing the requirement for real-time analytics and stream processing to facilitate prompt and efficient responses by businesses and applications. It delves into different scenarios where stream processing is applicable, including real-time monitoring and predictive analytics. The article introduces incremental ETL, online machine learning, and structured streaming as remedies for managing real-time data to make proactive decisions and enhance system performance. Structured streaming, which is based on Spark SQL, provides scalable and resilient features along with APIs in various programming languages for efficient stream processing. It is enhanced by Delta Lake, a storage layer that allows for optimized data querying and modifications through schema evolution and stream-stream joins. The combination of Structured Streaming and Delta Lake enables effective data pipeline management and automated rate-limiting tuning, resulting in a seamless integration of batch and stream processing. The article also examines the difficulties connected to stream processing, such as dealing with out-of-order or outdated data and achieving scalability without sacrificing performance. Delta Lake enhances this procedure by being compatible with both append and complete output modes, thus providing flexibility in managing stream data.

REFERENCES

[1]. T. Kolajo, O. Daramola, and A. Adebiyi, "Big data stream analysis: a systematic literature review," Journal of Big Data, vol. 6, no. 1, Jun. 2019, doi: https://doi.org/10.1186/s40537-019-0210-7.

[2]. H. Nasiri, S. Nasehi, and M. Goudarzi, "Evaluation of distributed stream processing frameworks for IoT applications in Smart Cities," Journal of Big Data, vol. 6, no. 1, Jun. 2019, doi: https://doi.org/10.1186/s40537-019-0215-2.

[3]. "6. Batch Is a Special Case of Streaming - Introduction to Apache Flink [Book]," www.oreilly.com. https://www.oreilly.com/library/view/introduction-to-apache/9781491977132/ch06.html

[4]. D. Shi, J. Zurada, W. Karwowski, J. Guan, and E. Çakıt, "Batch and data streaming classification models for detecting adverse events and understanding the influencing factors," Engineering Applications of Artificial Intelligence, vol. 85, pp. 72–84, Oct. 2019, doi: https://doi.org/10.1016/j.engappai.2019.05.006.

[5]. T. Kolajo, O. Daramola, and A. Adebiyi, "Big data stream analysis: a systematic literature review," Journal of Big Data, vol. 6, no. 1, Jun. 2019, doi: https://doi.org/10.1186/s40537-019-0210-7.

[6]. A. Saxena, "Spark Streaming vs. Structured Streaming," blog.knoldus.com, Feb. 28, 2019. https://blog.knoldus.com/spark-streaming-vs-structured-streaming

[7]. "Structured Streaming Programming Guide - Spark 2.1.2 Documentation," spark.apache.org. https://spark.apache.org/docs/2.1.2/structured-streaming-programming-guide.html

[8]. "How Tilting Point Does Streaming Ingestion into Delta Lake," Databricks, May 10, 2019. https://www.databricks.com/blog/2019/05/10/how-tilting-point-does-streaming-ingestion-into-delta-lake.html