International Journal on Science and Technology (IJSAT)
E-ISSN: 2229-7677 ● Website: www.ijsat.org ● Email: editor@ijsat.org

Incremental Processing For Handling Late-Arriving Data in Batch Processing

Arjun Reddy Lingala

arjunreddy.lingala@gmail.com

Abstract

Batch processing systems often struggle with the challenge of handling late-arriving data [5], leading to incon- sistencies in analytical results and unnecessary computational overhead. This paper introduces an incremental processing [10] that efficiently incorporates late data into batch datasets by reducing user overhead of maintenance, avoid mistakes from users and saving computational overhead. Some of the data arriving into data warehouse often gets delayed due to multiple upstream issues like network outages, upstream delays, fixes from data reconciliation. Late-arriving data presents significant challenges in batch and real-time data processing environments which impact data accuracy [3], system efficiency, and overall analytics reliability. Unaccounted late arriving data can lead to incomplete and inaccurate analytical results and the dashboards generated from them represent incorrect metrics. It often re- quires late arriving data to be caught or raised with an alert and then the user who owns the ETL has to re-run the batch pipelines that are within the time or date range of the arrived data. This approach significantly reduces the computational over- head of batch reprocessing while ensuring data consistency and completeness. The proposed framework introduces a real-time detection mechanism that continuously monitors incoming data and identifies late records by comparing timestamps with pre- existing batch data. Once late data is detected, a targeted backfill strategy is applied, ensuring that only the affected time partitions starting from the hour of late data arrival until the current processing period are recomputed in sequential approach for a dataset that depends on order and historical information and only delta from the time of late arriving data is actioned upon for a dataset that doesn't depend on the historical data. This selective reprocessing minimizes redundant computations and optimizes system performance with fault tolerance and scalability handling large volumes of late-arriving data in distributed environments.

Keywords: Batch Processing, Late Data, Incremental processing, Distributed systems, ETL, Real-time, Intra-day pipelines, Scalability, Fault tolerance, Backfilling, Dashboards, Metrics

I. INTRODUCTION

Batch processing plays a crucial role in large-scale data engineering, serving as the backbone for analytics, machine learning workflows, and ETL (Extract, Transform, Load) processes. Conventional batch systems operate under the assumption that all required data is available at the time of execution. However, real-world scenarios frequently encounter late-arriving data due to network latency, system failures, or out-of-order event occurrences. Such delays create substantial issues, including data inconsistency, inefficient computational resource usage, and increased latency and incorrect metric

representation in datasets and dashboards. The traditional solution of reprocessing entire datasets to accommodate late data is resource-intensive and impractical at scale. To mitigate these inefficiencies, we propose an incremental processing framework [10] that automatically detects late-arriving data [5], selectively backfills missing records from the hour of arrival to the present, based on the state of the dataset (whether it depends on historical information or not) for optimized processing.

Managing late-arriving data [5] in batch processing introduces several critical challenges. Late-arriving records can result in incomplete or erroneous aggregations, leading to incorrect analytical conclusions. Reprocessing entire datasets in response to late data arrivals consumes significant CPU and memory resources. Efficiently integrating late data without introducing delays in ongoing batch jobs is essential. Stateful datasets necessitate complex updates and state modifications as the late arriving data can impact the final state in stateful datasets, whereas stateless datasets may require simpler adjust-ments. Implementing intelligent strategies for integrating late- arriving data from its arrival time to the present state without excessive recomputation. In this paper, we address these chal- lenges by using a monitoring mechanism dynamically identifies and flags late-arriving records for targeted backfilling, instead of reprocessing full datasets, our framework updates only the affected time window, from the hour of arrival to the current state, applying different strategies for stateful and stateless datasets to ensure efficient processing.

II. Challenges with Late-Arriving Data

Late-arriving data presents significant challenges in batch and real-time data processing environments. These challenges can impact data accuracy, system efficiency, and overall analytics reliability. When late-arriving data [5] is not accounted for, it can lead to incomplete or inaccurate analytical results. Reports and dashboards generated from incomplete data may mislead decision-makers. In cases of financial transactions, fraud detection, or real-time monitoring, even minor inconsistencies can have serious consequences. Traditional batch systems often require reprocessing the entire dataset to incur porate late-arriving data, which is computationally expensive. This leads to unnecessary resource consumption, increasing CPU, memory, and storage costs. Full reprocessing also delays updates, making data pipelines less efficient. When new data arrives late, it may fall outside the pre-defined processing window, leading to incorrect aggregations. Adjusting or re- calculating windowed results requires complex handling and additional computations. Many ML models and AI-driven analytics rely on timely and complete data for training and inference. If late-arriving data [5] is excluded or processed in- correctly, it can skew model predictions and degrade accuracy. Systems need to be fault-tolerant to handle scenarios where late data is caused by network failures, processing delays, or external API slowdowns. Implementing robust checkpointing, replay mechanisms, and state management is necessary to recover from failures while correctly integrating late data. Late-arriving data requires additional logic in ETL pipelines. Handling out-of-order data properly increases the complexity of data governance, tracking dependencies, and maintaining lineage. Some systems must merge streaming and batch work-loads to ensure completeness, adding architectural challenges.

III. Incremental Framework

The proposed framework for handling late-arriving data [5] in batch processing systems is designed with flexibility, efficiency, and scalability in mind. The core principle of this framework is to introduce a modular approach for the automatic detection of late data, the efficient backfilling of missing records, and the tailored handling of stateful versus stateless datasets. The framework can be

implemented in various distributed processing systems and is specifically engineered to handle the challenges posed by the time-sensitive nature of modern data pipelines.

A. Late Arriving Data Detection

One of the key challenges of late-arriving data is its timely detection, as systems must accurately recognize when incoming data falls outside its expected arrival time window. Our framework solves this by implementing a real-time event- based detection system that compares incoming data times- tamps with predefined time windows, which are typically defined based on the periodicity of the batch (e.g., hourly, daily). Each data record includes metadata containing the timestamp of when the record was generated, allowing the system to determine its expected arrival time. Incoming data contains two types of timestamps – one is the timestamp when the event is created which is creation time and the second one is the timestamp when the event is processed which is processed time. Users needs to define which timestamp is considered for the definition of late arriving data. In addition to this, users also needs to define the grace period which explains of how many units (hour, day) are we considering for the late-arriving data. For example, if the current time is X, and the grace time is defined as 10 units, then the late- arrived data with creation timestamp anything less than X- 10 will be ignored for processing as this falls outside grace period window. Late-arriving data is defined as any record that is received after the expected time window has closed but within a pre-defined grace period. This grace period helps to accommodate minor delays due to network or system latencies without classifying data as late unnecessarily. After the data is arrived, the timestamp of the incoming data is compared to the time window in which it was expected. If the data's timestamp indicates it falls within an outdated time window, it is flagged as late. The framework considers the grace period allowed for late data. If the data arrives after this grace period, it may be rejected or handled. Once late data is detected, an event is generated, triggering the backfilling mechanism to process the missing data appropriately.

B. Backfill

Once late-arriving data [5] is detected, it triggers the back- filling process, a crucial part of the framework. Backfilling refers to the action of reprocessing or updating the previously completed records to include the late data and ensure that the results reflect the most current and accurate information. The backfilling process operates incrementally, meaning that only the late-arriving data [5] is processed and integrated, without requiring the re-execution of the entire batch. This targeted approach not only saves computational resources but also reduces the overall processing time, which is crucial for maintaining the efficiency of the system. The backfilling process starts when late data arrives, the system identifies the specific time window it corresponds to. For both stateful and stateless datasets, some derived metrics or aggregations may be impacted by late data. For instance, late data may affect rolling averages, sums, or other aggregate values that depend on the time window. Our framework intelligently recalculates these derived metrics only for the affected time windows and affected data points, ensuring that the overall impact on system performance remains minimal.

C. Stateful vs Stateless

A defining feature of the proposed framework is its distinction between stateful and stateless datasets. Each type of dataset has different processing requirements when handling late-arriving data. Our framework is specifically designed to accommodate these differences by applying different strategies to ensure consistency and correctness in both cases.

1) Stateless Datasets: Stateless datasets do not maintain any historical context or accumulated state. Examples include time-series logs, sensor event records, and non-cumulative data where each entry is independent. Handling of late-arriving data
[5] for stateless datasets is much simpler. Late-arriving data is inserted directly into the dataset at the appropriate time window or timestamp. As these datasets do not depend on historical context, no recalculation or state maintenance is required. While individual records are independent, aggregations based on these records (e.g., sum, average, or count) may be affected by late data. The system selectively recalculates the affected aggregation windows, ensuring that only the impacted parts of the data are recomputed optimizing the backfilling process.

2) Stateful Datasets: Stateful datasets maintain contextual information across time, meaning that the system tracks changes over time and updates the dataset's state. Examples of stateful datasets include financial transactions or sensor data. The late data is not just inserted into the dataset but also requires the updating of the system's historical state. For instance, if the dataset maintains an aggregate such as a running total, the framework must recalculate the running total by taking into account the new data point. In some cases, the impact of late-arriving data may extend beyond the immediate time window. For instance, a user's transaction history might need to be revisited to adjust previously recorded balances. The framework handles this by tracing back the affected records and applying the updates where necessary. To efficiently manage and update the state, we use an incremental state maintenance model [10]. This model ensures that only those parts of the state that are directly affected by the late data are recalculated. By using versioned data models, the system can track and maintain historical versions of states without needing to recompute the entire state history. This can be easily achieved by using modern table formats in distributed systems like Apache Iceberg [1] which maintains the table updates info through versioning.

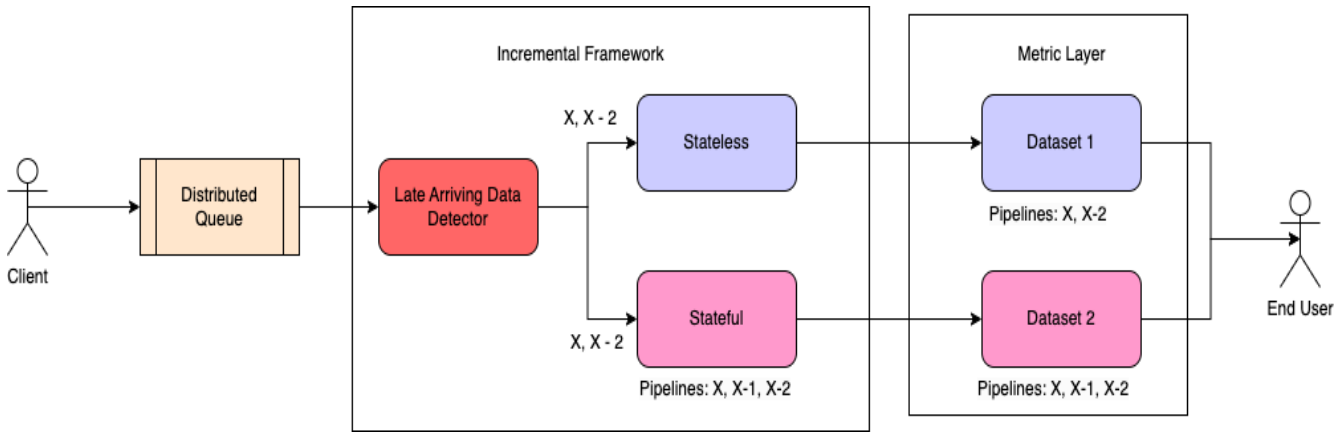

Fig. 1. Incremental Processing Framework

IV. ARCHITECTURE

The architecture of the proposed Incremental Processing Framework is built to ensure scalability, flexibility, and efficiency in handling late-arriving data in batch processing systems. It is designed to handle large volumes of data, perform real-time detection of late-arriving data, and apply efficient backfilling techniques to maintain consistency across stateful and stateless datasets.

The Late-Arrival detection and classification Layer is responsible for identifying whether the

incoming data is late. The system employs time-based validation to determine whether data arrives within the expected time window or if it is delayed. This layer involves multiple components that work together to handle the detection and classification of late data. Time Window Monitoring is a mechanism that tracks the time of arrival for each batch and event. The expected time window is pre-configured based on the specific needs of the system. Any data received outside of this window is flagged as late. When data arrives past the expected time window, it is processed by the Late Data Detection Engine, which evaluates whether the data can still be processed under a predefined grace period. If it falls within this period, the data is considered valid for backfilling. The engine classifies late- arriving data into categories called minor delay which is the data arrives within the permissible grace period, major delay which is the data arrives outside the grace period, possibly requiring reprocessing of larger parts of the dataset or manual intervention, and non-late data which is the data that arrives on time and doesn't need any additional handling.

The backfilling and data insertion Layer plays a crucial role in ensuring that late-arriving data is seamlessly integrated into the existing data set without causing discrepancies. Late Data Insertion engine inserts the late data into the appropriate time window. The backfilling process is handled incrementally by determining which records need to be updated based on the time window affected by the late data. It is important that this insertion doesn't overwrite existing data but adds the late records to the correct chronological spot. It is required to do proper audits using data quality checks before and after the backfill process is done to make sure that it doesn't impact the overall metrics incorrectly. The processing and aggregation layer responsible for handling the processing of both stateful and stateless datasets. For stateful data, it is responsible for handling datasets that require historical context, where the data is interdependent. The processor ensures that late-arriving data updates the current state appropriately and maintains the chronological integrity of the dataset. When late data is inserted into a stateful dataset, it can affect previous records or aggregates. This component recalculates historical data incre- mentally, ensuring that stateful information, such as balances or time-series trends, remains accurate. For stateless data, late data is directly inserted into the dataset with no need for previous record recalculations and this component recalculates only the impacted time window's aggregates. It ensures that the data remains consistent without needing to recalculate the entire dataset. To ensure transparency, traceability, and ease of debugging, the framework includes an Event Notification and Logging System. The system generates real-time alerts whenever late data is detected, backfilled, or aggregated. These notifications are essential for monitoring the system's state and ensuring timely actions are taken for manual intervention if necessary. Every step of the backfilling and processing pipeline is logged, including the detection of late data, backfilling operations, and the recalculation of aggregates. This log provides a detailed trail that can be used for troubleshooting and auditing purposes.

V. CONCLUSION

In this paper, we have proposed an incremental processing framework [10] designed to efficiently handle late-arriving data in batch processing systems. The framework automatically detects late data, backfills it from the point of arrival until the current moment, and processes it in a manner that distinguishes between stateful and stateless datasets. With the increasing complexity of data-driven systems and the growing importance of real-time data streams, effectively managing late-arriving data has become a critical challenge. The frame- work introduced in this study addresses this issue by incorporating advanced techniques such as time-window validation, delta-based backfilling, and

incremental processing to ensure timely and accurate data handling. The central objective of this work was to create a solution that minimizes the impact of late-arriving data on ongoing batch processing operations while preserving the consistency and integrity of the datasets. For stateful datasets, the system leverages a robust handling mechanism that updates running totals and aggregates only when necessary, thus minimizing unnecessary recalculations.

For stateless datasets, the framework allows for simpler, direct updates to the dataset with minimal computational overhead. This tailored approach ensures that each dataset type is man- aged in the most efficient way possible, avoiding redundant operations and reducing system load. While the proposed framework shows great potential, there are several areas for future exploration. One promising direction is the incorporation of machine learning techniques for more adaptive and intelligent handling of late data. By analyzing historical pat- terns of data arrival, the system could dynamically adjust grace periods or trigger more advanced backfilling strategies, thereby optimizing data processing further. Another possible enhancement is the integration of real-time data validation mechanisms to automatically identify anomalies or inconsistencies in the incoming late data. In conclusion, this framework addresses the critical challenge of handling late-arriving data in batch processing systems with efficiency, scalability, and accuracy. By minimizing disruptions to ongoing operations and ensuring data consistency, it enhances the reliability and performance of batch processing systems, enabling better decision-making and more effective data-driven strategies across industries.

REFERENCES

[1] R. Blue, D. Weeks, and J. Turnbull, ”Apache Iceberg: A Mod- ern Table Format for Big Data,” in Proc. IEEE 36th Int. Conf. Data Eng. (ICDE), Dallas, TX, USA, 2020, pp. 1645–1658. doi: 10.1109/ICDE48307.2020.00147.

[2] K. Shvachko, H. Kuang, S. Radia and R. Chansler, ”The Hadoop Distributed File System,” 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), Incline Village, NV, USA, 2010, pp. 1-10, doi: 10.1109/MSST.2010.5496972.

[3] A. Katsifodimos and S. Schelter, ”Apache Flink: Stream Analytics at Scale,” 2016 IEEE International Conference on Cloud Engineer- ing Workshop (IC2EW), Berlin, Germany, 2016, pp. 193-193, doi: 10.1109/IC2EW.2016.56.

[4] A. S. A. de Oliveira, M. A. S. Santos, R. A. C. Ferreira, and R. T. de Sousa Jr., ”A Survey of Distributed Data Stream Processing Frameworks,” IEEE Communications Surveys and Tutorials, vol. 20, no. 2, pp. 1034–1058, Secondquarter 2018.

[5] T. Akidau et al., ”The Dataflow Model: A Practical Approach to Balancing Correctness, Latency, and Cost in Massive-Scale, Unbounded, Out-of-Order Data Processing,” Proc. VLDB Endow., vol. 8, no. 12, pp. 1792–1803, Aug. 2015. doi: 10.14778/2824032.2824076.

[6] ”Apache Flink: Stream and Batch Processing in a Single Engine,” IEEE Data Eng. Bull., vol. 38, no. 4, pp. 28–38, Dec. 2015.

[7] ”Big Data: Principles and Best Practices of Scalable Realtime Data Systems,” Manning Publications, 2015. [Online]. Available: https://www.manning.com/books/big-data.

[8] ”C-Store: A Column-oriented DBMS,” in Proc. 31st Int. Conf. Very Large Data Bases (VLDB), 2005, pp. 553–564.

[9] ”The Stratosphere Platform for Big Data Analytics,” VLDB J., vol. 23, no. 6, pp. 939–964, Dec. 2014. doi: 10.1007/s00778-014-0357-y.

[10] "Incremental Checkpointing for Stateful Stream Processing Systems," in Proc. IEEE Int. Conf. Data Eng. (ICDE), 2017, pp. 1359–1370. doi: 10.1109/ICDE.2017.184.

[11] "Twitter Heron: Stream Processing at Scale," in Proc. ACM SIG- MOD Int. Conf. Manage. Data (SIGMOD), 2015, pp. 239–250. doi: 10.1145/2723372.2742788.