International Journal of Scientific Research in Modern Science and Technology
ISSN: 2583 -7605 (Online)
© IJSRMST | Vol. 3 | Issue 6 | June 2024
Available online at: https://ijsrmst.com/
DOI: https://doi.org/10.59828/ijsrmst.v3i6.215

Real-time data analytics in distributed systems

Anju Santosh Yedatkar

Assistant Professor, Department of Computer Science, Sarhad College of Arts Commerce and Science, Katraj, Pune

Email: anjuyedatkar@sarhad.in

ABSTRACT

Real-time data analytics involves the processing and analysis of data as it arrives, delivering immediate insights that are crucial for time-sensitive applications. This research explores the platforms and techniques necessary for supporting real-time analytics, extending beyond traditional Event Processing Systems (EPS) to include broader big data contexts that integrate both 'data at rest' and 'data in motion' solutions. A detailed case study is presented, showcasing the application of the Event Swarm complex event processing engine in addressing financial analytics challenges. The study identifies key research challenges in the field, emphasizing the need for innovative approaches and algorithms to enhance real-time data filtering, exploration, statistical analysis, and machine learning.

Keywords: real-time analytics, complex event processing, streaming, event processing, advanced analytics, data analysis, machine learning, finance, Event Swarm.

Introduction:

This paper explores the field of real-time analytics, emphasizing the importance of addressing the velocity aspect of big data. As the proliferation of data continues to accelerate, Instantaneous data processing and analysis has become essential for both business and social applications. Often referred to as "Fast Data," this high-velocity data is crucial for real-time interactions and serves as an early warning system for potential issues. However, despite the rapid growth of data, large enterprises have historically focused more on managing data volume and variety than on the challenges posed by high-velocity data.

Building upon the fundamentals of distributed computing, event processing, and machine learning, real-time analytics extends the concepts of real-time applications from before the Internet era to the current era, marked by the emergence of cloud computing, the expansion of data sources like social media and mobile devices, and an increasing demand for data insights. Unlike earlier distributed systems, which were designed for specialized audiences using proprietary infrastructure, today's real-time analytics must handle continuous data streams from diverse sources, posing significant computational and resource challenges.

This paper discusses the evolution of real-time infrastructure, tools, and analytics techniques, positioning them within the broader context of traditional stored-data analytics and event-driven architectures. It

introduces a hierarchical abstraction model, termed the "analytics stack for big data," which serves as a framework for discussing the various concepts, technologies, and principles covered.

Key characteristics of real-time systems, rooted in early distributed system developments, are examined, setting the stage for an exploration of current solutions and challenges

Characteristics of Real-Time Systems:

Real-Time Systems: Real-time systems process events as they occur within a specified time interval, often measured in milliseconds, microseconds, or nanoseconds. Timeliness is essential for correctness in real-time systems, particularly in safety-critical applications. However, in big data analytics, "real-time" often refers to near real-time processing, where data is processed as it arrives.

Low Latency: Latency is the delay between an event and the start of its processing. Real-time systems require low latency for timely responses. Strategies to reduce latency include in-memory processing, flash storage, incremental evaluation, parallel processing, and anticipatory data fetching.

High Availability: High availability guarantees that a system will be able to operate as needed. This is important for real-time systems since processing that is delayed can cause events to be missed. Data replication, redundant processing, and distributed processing are high availability strategies.

Horizontal Scalability: Horizontal scalability allows the addition of servers to an existing pool to improve performance or increase capacity, which is vital for handling variable data ingress rates.

Real-Time Processing Concepts and Platforms:

Event and Event Processing: An event signifies an occurrence in a system's environment, and event processing involves operations on these events. Event-driven architectures are common in real-time systems, leveraging the availability and scalability of distributed infrastructure.

Event Stream Processing and Data Stream Processing: These processes involve the continuous flow of data, often from sensors or other applications, and their processing in real-time to detect patterns or generate alerts.

Complex Event Processing (CEP): CEP involves complex computations over multiple events, identifying meaningful patterns and relationships between them.

Data Stream Processing Platforms:

Hadoop Ecosystem: A family of projects supporting large-scale data processing, primarily designed for batch processing but can be linked with real-time systems like Storm for real-time analytics.

Apache Spark: A distributed computing framework that supports real-time processing through Spark Streaming, which processes data in micro-batches. Storm: An unbounded stream of data processing system that uses bolts (data processors) and spouts (data sources) for distributed real-time computation.

Kafka: A high-throughput, low-latency platform for handling real-time data feeds, commonly used for message processing.

Flume: A service for collecting, aggregating, and moving large amounts of log data, suitable for data ingestion and simple event processing.

Amazon Kinesis: A cloud-based service for real-time data processing, capable of capturing and storing large data streams, and integrating with systems like Storm for scalable stream processing.

1. **Query-Based Event Processing Systems (EPS) Description:** These systems typically use an Event Processing Language (EPL) similar to SQL, extended to handle continuous streams of event data.

Functionality: Continuous Queries: These are persistent and continuously applied to data streams, in contrast to standard SQL queries. Sliding windows are a key component that allows event streams to be divided into manageable portions for analysis. Several systems that use time- or tuple-driven models implement this.

As an example: CQL: Uses a time-driven paradigm to reevaluate windows at the end of each time step. With each new data tuple, Stream SQL reevaluates windows using a tuple-driven approach. Microsoft's cloud-based Azure Stream Analytics platform enables real-time streaming analytics through sophisticated event processing via a language akin to SQL.

2. **Rule-Oriented Event Processing Systems Description:** These systems are better suited for describing higher-level business events using conditional logic.

Types:

Production Rules: Originating from expert systems, these rules follow a format of "if Condition then assert Conclusion/Action" and are often used in domains like medicine or telecommunications. Forward-Chaining: A common operational semantics in these systems where rules react to changes in conditions.

Limitations: Traditional production rules struggle with complex event patterns and may lack declarative semantics, leading to issues like termination problems.

ECA (Event-Condition-Action) Rules: ECA rules trigger actions based on specific events and conditions. These rules are commonly used in active databases and can support composite events.

Applications: ECA rules are used in systems like IBM's Info Sphere Streams and can be applied to real-time business logic or database integrity monitoring.

3. Programmatic Event Processing Systems

Description: These systems offer a more flexible and extensive framework for handling complex event streams, often allowing for custom programming.

Example: EventSwarm: A Java-based framework that provides predefined abstractions and pattern components for event processing.

Capabilities: Filtering, transforming, and aggregating event streams. Incremental updates on sliding windows.

Common Operations Across EPS:

Event Expressions: Basic building blocks for specifying matching criteria for events.

Filter: Reduces the events processed by selecting relevant subsets.

Transformation: Alters event data, including translation, splitting, aggregation, and composition.

Event Pattern Detection: Identifies high-level patterns by correlating, aggregating, or abstracting events.

Key Points: Flexibility and Complexity: While query-based EPSs are straightforward and suitable for simple continuous queries, rule-based and programmatic EPSs offer more flexibility and power but may introduce complexity, especially in expressing intricate event patterns.

Real-Time Analytics: The ultimate goal of these platforms is to support real-time decision-making by processing and analyzing event streams as they occur, which is crucial for applications like financial trading, healthcare monitoring, and business process management.

Data Analysis in General

Purpose: Data analysis aims to convert raw data into useful information to answer questions, suggest conclusions, and support decision-making.

Phases of Data Analysis: Data Collection: Gathering relevant data from sources necessary for answering research questions.

Data Transformation: Preparing data for analysis through standardization or normalization to ensure consistency in format. Analyzing data statistically involves finding patterns and trends. There are various steps involved, such as:

Information Nature Description: Knowing the fundamental properties of the information. Building models to explain the relationship between the data and the underlying population is known as data modeling.

Model validation: Assuring the model's accuracy. Making predictions with the model is known as predictive analytics.

Instruments for Processing Data: Pre-processing Tools: Data Wrangler and OpenRefine are two examples. Database management solutions include NoSQL systems like Cassandra and MongoDB and SQL-based systems like MySQL and PostgreSQL. R, SAS, Stata, and SPSS are statistical analysis tools.

Analyzing Data for Applications in Streams

Key Differences in Stream Analytics: Dynamic Data Sets: Unlike static historical data, stream data changes over time as new data arrives and old data is removed. Sliding Window Abstraction: Used to continuously update computations over streams of data.

Simple Calculations: Calculations like minima, maxima, mean, averages, and standard deviations are updated incrementally with each new event.

Complex Calculations: More involved algorithms, such as multivariable regressions, may require specialized techniques like sampling (e.g., sliding window reservoir sampling, biased reservoir sampling).

Handling Out-of-Order Events: Due to network delays, events may arrive out of order, requiring mechanisms like buffering, recalculation, or methods that accommodate out-of-order events.

Time Series Analysis: Many real-time analytics tasks in stream processing resemble time series analysis, but require modification to suit the dynamic nature of streaming data. Real-Time Analytics in the Finance Domain

Context: The finance domain has seen significant changes due to the rise of high-frequency algorithmic trading, which generates large volumes of real-time "tick-by-tick" data.

Data Providers: Major providers include Thomson Reuters, Bloomberg, and WRDS, which offer real-time and historical financial market data.

Challenges: Real-time data quality control is critical. Issues like duplicate events or incorrect data can lead to erroneous analysis.

Case Study: The case study simulates real-time data processing using historical data from Thomson Reuters Tick History.

Scenario 1: Deals with duplicate dividend announcement events, which call for certain processing and detection methods.

Important lessons learned: Data analysis is a methodical procedure that includes multiple steps, starting with gathering and ending with statistical analysis. Stream data analytics presents unique challenges, including handling dynamic data sets, out-of-order events, and the need for continuous updates. In finance, real-time analytics is critical for maintaining data quality and ensuring accurate analysis, especially in high-frequency trading environments.

CEP Application Overview

1. Introduction

The CEP application discussed is designed to handle complex event processing for financial market data. It integrates two main components: Financial User Front-End, Back-End CEP Engine

2. System Architecture

Financial User Front-End: Functionality: Allows financial data analysts to select and define rules for processing financial market data streams.

Interaction: Users interact with the Front-End to specify which event patterns they want to monitor.

Back-End CEP Engine: Functionality: Detects and processes event patterns based on the rules defined in the Front-End.

Technology: Implemented using the EventSwarm framework.

Communication: Interface: RESTful API for communication between the Front-End and CEP engine.

Data Format: Results are encoded in JSON and sent back to the Front-End using HTTP POST requests.

3. Event Swarm Framework

Implementation: Platform: Ruby on Rails on the JRuby platform.

Pattern Definition: Patterns are coded in Ruby and are used to build a CEP graph.

Performance: Pattern matching is executed in Java for efficiency, while some elements are handled in Ruby.

Processing Graphs: Components: Includes sliding windows, filters, splitters (powersets), and abstractions.

Abstractions: Maintain statistical measures such as sum, mean, variance, and standard deviation.

Event Management: Processing: Events are added to and removed from the stream by nodes, which may choose to ignore removals (e.g., sliding windows).

4. Deployment and Testing

Steps:

1. Rule Specification: Financial analysts describe event pattern rules in natural language.
2. Pattern Implementation: IT experts code these rules into the EventSwarm system.
3. Rule Execution: Researchers select rules via the Front-End, which sends HTTP GET requests to EventSwarm.
4. Result Handling: Detected patterns are returned in JSON format and processed by the Front-End.

5. Benefits

API Integration: Simple and user-friendly API for developers.

Performance: High-speed processing, handling over 10,000 events per second.

Deployment Speed: Quick implementation of new event patterns, typically under one day for five patterns.

Output Structure: Well-structured JSON format for easy parsing and analysis.

6. Limitations

Pattern Definition: Lack of a user-friendly GUI or API for defining event patterns, requiring IT expertise.

Communication Challenges: Effective communication between researchers and IT experts is crucial; misunderstandings can be difficult to diagnose.

7. Conclusion

The implementation of the CEP application using EventSwarm demonstrates effective real-time processing of financial data with high performance and rapid deployment capabilities. The system's simplicity and efficiency are contrasted with challenges related to pattern definition and communication, highlighting areas for potential improvement.

8. Future Directions

Future improvements could focus on enhancing the user interface for defining event patterns and refining communication processes to reduce dependency on IT experts

Future Research Challenges in Real-Time Analytics

1. Refining Analytics Techniques for Streaming Data

Challenge:

Objective: Improve and adapt existing analytics methods, including statistical and machine learning techniques, to effectively handle streaming data.

Applications: Precision Monitoring: Enhancing the accuracy of monitoring systems for business, quality, and security issues.

Data Insights: Developing more timely and relevant insights from data streams.

Predictive Analytics: Improving the outcomes of predictive analytics by applying refined techniques to real-time data.

Research Directions:

Algorithm Adaptation: Modify statistical and machine learning algorithms to accommodate the continuous and evolving nature of streaming data.

Real-Time Data Processing: Explore techniques for processing and analyzing data in real-time to generate actionable insights quickly.

2. Standardization of Formats for Interoperability

Challenge:

Objective: Establish standards for representing and exchanging event abstractions, such as events and event patterns, to facilitate interoperability across various systems.

Issues: Lack of Standard Formats: Current absence of standardized formats for event representation impedes integration and communication between different real-time analytics systems.

Model Representation: Need for new models and standards that accurately represent streaming data abstractions.

Research Directions:

Model Development: Propose and develop standard models for event representation and event pattern abstraction.

Interoperability Solutions: Create standards that ensure seamless integration between streaming data platforms and stored data systems.

3. Developing User-Friendly Solutions

Challenge:

Objective: Create solutions that align with existing analysis methods and practices in specific application domains, making it easier for users to define and manage analytics abstractions.

Issues: Domain-Specific Models: Data analysts currently struggle to define event patterns and analytics abstractions without specific tools or models.

Model-Driven Approaches: Need for models that drive analytics infrastructure in a domain-specific manner.

Research Directions:

Domain-Specific Analysis Models: Explore the development of models tailored to specific application domains that guide the analytics infrastructure.

Approaches to Consider: Domain-Specific Languages (DSLs): Investigate whether DSLs could provide a structured way for users to define and manage analytics abstractions.

Semantic Technologies: Evaluate the potential of semantic technologies to enhance the definition and application of analytics models

References

[1] Milosevic, Zoran, Chen, Weisi, Berry, A., Rabhi, Fethi. (2016). *Real-Time Analytics*. doi: 10.1016/B978-0-12-805394-2.00002-7.

[2] S. Ashraf, Y. M. Afify, R. Ismail. (2022). *Big Data for Real-Time Processing on Streaming Data: State-of-the-art and Future Challenges*. In: Proceedings of the 2022 International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME), Maldives, Maldives, pp. 1-8. doi: 10.1109/ICECCME55909.2022.9987770.

[3] Akanbi, Adeyinka, Masinde, Muthoni. (2020). *A Distributed Stream Processing Middleware Framework for Real-Time Analysis of Heterogeneous Data on Big Data Platform: Case of Environmental Monitoring*. Sensors, 20(11), 3166. doi: 10.3390/s20113166.

Cite this Article:

***Anju Santosh Yedatkar**, **"Real-time data analytics in distributed systems"** International Journal of Scientific Research in Modern Science and Technology (IJSRMST), ISSN: 2583-7605 (Online), Volume 3, Issue 6, pp. 09-16, June 2024.*

Journal URL: https://ijsrmst.com/

DOI: https://doi.org/10.59828/ijsrmst.v3i6.215.