REVIEW ARTICLE

STREAM ALGEBRA FOR BIG DATA ANALYSIS

Tran Van Lang

HCMC University of Foreign Languages - Information Technology

langtv@huflit .edu.vn

ABSTRACT — This article presents an overview of Stream Algebra, a research field that has emerged after the birth of Relational Algebra used in Database Management Systems (DBMS). When Stream Data appeared and needed to be processed in real-time, the role of Stream Algebra became evident. Since the rise of Big Data, this field has received attention in recent years, the role of Stream Algebra has been further demonstrated. In addition, the article discusses the analysis of big data based on the approach of Stream Algebra, thereby contributing to determining the research direction in the era of Data Science for young researchers, graduate students who want to find a challenging research direction. Special focus on analysis to present some open issues in research on application of Stream Algebra. The article also presents some Frameworks utilizing Stream Algebra to help stream data management effectively for quick use in research and implementation.

Keywords — Big Data, Data Science, Relational Algebra, DBMS.

I. INTRODUCE

Stream Algebra is a field of study in mathematics and computer science that deals with the modeling and processing of streams of data **[1]**; especially in Big Data analytics, where data is processed and analyzed in real or near real time. A data stream can be understood as a sequence of elements (*data, events, signals*) generated over time, usually in temporal and spatial order. Stream Algebra is fundamentally different from traditional Relational Algebra, which is the foundation of conventional Database Management Systems (*DBMS*). Stream Algebra provides a set of operations and rules for processing, transforming, and analyzing data streams. Operations in Stream Algebra are typically designed to:

- Filtering: Filter out elements that meet a specific condition.
- Transformation: Converting data from one form to another.
- Aggregation: Summarize or calculate a value that represents all or part of a stream, such as an average, sum, or maximum value.
- Join/Merging: Combine multiple data streams together.
- Windowing: Divide the data stream into small blocks or windows based on time or number of elements.

Stream Algebra has a deep connection with Mathematics and Formal Language Theory; since data streams are not just discrete entities but also structured and sequential, it involves mathematical concepts of sequences, mappings, and formal languages. Specifically:

- A data stream can be viewed as a function that maps from a set of indices (usually times) to a set of values. For example, a temperature stream $T(t)$ maps time t to the corresponding temperature.
- A data stream can be represented as an infinite or finite sequence of elements $S = (s_1, s_2, s_3, \dots)$.
- Stream operations (*like filtering, pooling*) correspond to sequence operations, like Fourier Transform or Convolution.
- When the data stream represents continuous signals, it belongs to the continuous function space. For example, an audio signal stream is a continuous function in the time domain.
- Algebra structures such as monoids, groups, and rings to describe operations. For example, concatenation of *two* streams can be viewed as an operation on a monoid.
- A stream can be viewed as a sequence of characters or words in a formal language. For example, a stream of keyboard events can be represented in a formal language where each event is a character.
- Operations on streams, such as filtering or combining, can be expressed using regular expressions. For example, a stream of events can be described using a regular expression to recognize specific patterns.
- Data streams can be modeled by Finite State Machines (FSM), each state corresponding to an element in the stream. For example, applications that process real-time event streams, such as network protocol analysis.
- When the data stream is more complex and needs a hierarchical structure, context-free grammars can be used to model the context-free grammars. For example, a source code stream read from a compiler can be parsed based on a context-free grammar.

Stream Algebra can be said to be an application at the intersection of mathematics and formal language theory, combining mathematical concepts (*sequences, mappings, integrals, abstract algebra*) with discrete models from formal language theory. In practice, this intersection appears in:

- Signal Processing: Combining continuous space (*topology*) and discrete structure.
- Natural Language Processing (*NLP*): Analysis of text streams or linguistic data.
- Software Engineering: Modeling event streams in software systems.

A recent study proposed an algebraic framework to enhance performance in processing computer vision data stream (**[2]**). This work studies the construction of computer vision systems using data stream concepts to process visual data streams. One challenge in building computer vision systems is that algorithms have different accuracy and speed depending on the content, type of data as well as the speed of incoming data. From there, how to tune these algorithms in large-scale systems for optimization. This work presents methods and algorithms to overcome these challenges and improve performance in building and optimizing large-scale computer vision systems. The approach is to describe an algebraic framework *to* mathematically describe computer vision pipelines when processing image and video streams. Thereby providing a formal and abstract method for optimizing computer vision pipelines.

In recent times, Stream Algebra has been put into practice to develop useful frameworks. For example, the article **[3]**on Microsoft Learn introduces windowing functions in Azure Stream Analytics, a real-time data analytics service from Microsoft that is used when processing data streams. In time-series data processing scenarios, performing operations on data in time-series windows is a common pattern. Stream Analytics supports windowing functions out of the box, allowing developers to create complex stream processing jobs with ease. There are five types of time-series windows to choose from:

- Tumbling Window: Divides the data stream into non-overlapping and non-duplicated time segments.
- Hopping Window: Similar to tumbling window but can overlap, allowing an event to belong to multiple windows.
- Sliding Window: Unlike tumbling and hopping, sliding windows only output results when the contents of the window change, that is, when an event enters or exits the window.
- Session Window: Groups events that occur close together in time, filtering out inactive periods.
- Snapshot Window: A snapshot window that groups events with the same timestamp.

These window functions are applied within the GROUP BY clause of a regular query in Stream Analytics jobs. All window operations output results at the end of the window. When starting a job in Stream Analytics, it is necessary to specify the start time of the job output, then the system automatically takes previous events in the input streams to output the first window at the specified time.

The article **[4]**focuses on techniques for displaying and analyzing results from windowing operations in two major stream processing frameworks, Kafka Streams and Flink SQL. It provides some illustrative examples and detailed explanations of how to implement these techniques in both Kafka Streams and Flink SQL, so that we can better visualize how to display and analyze results from windowing operations in event stream processing.

Article **[5]**on Timeplus analyzes the difference between Complex Event Processing (*CEP*) and Event Stream Processing (ESP), two important methods in real-time data processing. ESP focuses on processing and analyzing events as they occur, typically in chronological order. The main goal is to detect patterns or trends in continuous data streams, allowing for rapid response to important events. ESPs are often used in applications that require immediate response, such as system monitoring or financial trading. CEP goes further by combining and analyzing multiple events from different sources to identify complex patterns or causal relationships. It does not just look at single events but also looks for meaningful combinations of events, helping to detect potential business opportunities or threats. CEPs are often applied in areas such as business operations monitoring or sensor network management. The article also details that ESP processes single events or sequential event chains, while CEP focuses on detecting complex patterns by combining multiple events from different sources. In application, ESP is suitable for situations that require quick response to single events, while CEP is used to detect complex event patterns and causal relationships between them. Through the article, it helps organizations choose the right method for their real-time data processing needs, ensuring timely and effective response to important events.

Many frameworks implement Stream Algebra; however, many challenges remain in applying it to big data processing.

A brief introduction to Stream Algebra and recent research is presented above. The remainder of the paper includes a detailed introduction to Stream Algebra in Section II; Section III presents the application of Stream

Algebra in big data analysis. Section IV addresses some of the challenges that remain in exploiting Stream Algebra as the world moves towards using data to create better human-supporting applications; some discussions and conclusions are presented at the end of the paper.

II. STREAM ALGEBRA AND SOME RELATED ALGORITHMS

A. BIG DATA

Big Data is a term referring to large, complex, and diverse volumes of data that exceed the processing capabilities of traditional data management tools and systems. Big Data focuses not only on quantity but also requires consideration of the velocity, variety, and potential value of data. Big Data is often described through 5V characteristics:

- Volume: Very large amounts of data, measured in terabytes, petabytes, or exabytes. Examples: data from social media, IoT sensors, online transactions.
- Velocity: Data is generated and processed rapidly in real or near real time. Examples: financial transactions, fraud detection systems.
- Variety: Data comes in many different formats, from structured data like database tables, to semi-structured data like XML, and unstructured data like text, images, and videos.
- Veracity: Data can be inconsistent, inaccurate, or contain noise, requiring processing methods to ensure quality and accuracy.
- Value: The potential value of big data depends on its ability to extract useful information for Decision-making.

Big Data is streaming data that *is* generated continuously, often at high speed and in large volumes (*e.g. IoT sensors, financial transactions, system logs*), and needs to be processed immediately to extract information value. Big Data comes from various sources in modern life such as:

- Social Media: Data from Facebook, Twitter, YouTube, and other social media platforms.
- IoT devices: Factory measurement sensors, self-driving cars, smart devices.
- Trading system: Transaction logs from banks, e-commerce, stock markets.
- Scientific systems: Data from scientific, astronomical, medical, and genetic studies.

Big data is the foundation for many advances in various fields:

- Business Analytics: Provides information to optimize business processes and increase profits.
- Artificial Intelligence: Provides large amounts of data to train machine learning and AI models.
- Healthcare: Supports patient data analysis to improve healthcare quality.
- Security: Monitor networks, detect and prevent cyber attacks or illegal activities.

While big data offers huge potential, it also comes with many challenges:

- Storage: The huge volume of data requires distributed storage systems, such as Hadoop Distributed File System (*HDFS*).
- Processing: Optimization tools and methods like MapReduce, Apache Spark are needed to process data efficiently.
- Quality assurance: Data often contains a lot of noise or errors, requiring cleaning and preprocessing.
- Security: Ensure data is not misused or stolen during storage and processing.

B. STREAM ALGEBRA

Stream Algebra is a set of mathematical rules and operations designed to manipulate and transform stream data. The main components of Stream Algebra include:

- *A* set of records created over time.
- Time windows (*time windows*/*windowing*): Divide stream data into small segments for processing.

Stream operations: Includes operations such as mapping, filtering, joining, grouping, and aggregation. For example, there are operations: Filter to get records with values greater than a threshold; Join to combine two data streams based on a key; Aggregate to calculate the average or sum over a time window.

Stream Algebra and Relational Algebra are both mathematical systems for modeling and manipulating data, but serve different purposes and have fundamental differences in how they work, the data they process, and the operations involved. In general, the comparison between Stream Algebra and Relational Algebra is as shown in Table 1.

Table 1. *Comparison of Stream and Relational Algebra*

Criteria	Stream Algebra	Relational Algebra
Data type	Continuous, chronologically ordered data	Static data, stored as tables
Processing mechanism	Real-time or near-real-time processing	Batch processing
Time window	Support, time-based data segmentation	Not applicable
Basic math	map, filter, join, aggregation, sliding windows	select, project, join, union, intersection
Performance	Optimized for data processing speed	Optimized for data storage and querying
Application	Real-time analytics, IoT, anomaly detection	Static data storage and analysis

The detailed differences are shown through the following comparisons.

1. DATA NATURE
Stream Algebra:
Data is represented as a continuous stream (*streams*), which can be infinite, with elements arriving over time. Focuses on data that is real-time, ordered, and often related to a temporal context.

For example: Data from sensors, events in IoT systems, or financial transactions.

Relational Algebra :
Data is organized into relations (*tables*), which are finite sets of records. Focuses on static data, independent of order, with no concept of time.

For example: Database like customer table or transaction table.

2. PROCESSING MODEL
Stream Algebra:
Operations that process each element in a stream as it appears, often based on temporal or sequential context. Real-time or near-real-time processing requires high performance and low latency. Windowing is an important concept for grouping and processing continuous data.

Relational Algebra:
Operations are applied to the entire data set at once. Processing is done offline or on static data stored in a database. There is no concept of time or data windows.

3. MATH
Stream Algebra:
The common operations as mentioned above, are

- Filter: Filter elements according to conditions.
- Transform: Convert data into another form.
- Aggregate: Calculates the average, sum, or count on a data window.
- Windowing: Group data by time windows.
- Stream Join: Combines two or more data streams.

Relational Algebra:
Standard operations:

- Selection: Select records according to conditions.
- Projection: Select specific properties.
- Join: Combines two tables based on a common key.
- Union, Intersection , Difference: Set operations on tables .

4. APPLICATION
Stream Algebra:
Real-time data processing in distributed systems, IoT, and Complex Event Processing – CEP; such as Apache Flink, Kafka Streams, Flink SQL, Spark Streaming, Azure Stream Analytics, …

Relational Algebra:
Query static data in relational databases; such as MySQL, PostgreSQL, Oracle, …

5. MATHEMATICAL PROPERTIES
Stream Algebra:

Depends on timeline and concept of order. Data can change continuously, so operations need to ensure continuous calculation and update of results.

Relational Algebra:
Depends on sets and concepts in set algebra. There is no continuous change or update when performing operations.

C. SOME ALGORITHM ON STREAM ALGEBRA

Some related algorithms about operations on Stream Algebra:

- Filtering algorithm: Used to remove unnecessary data, for example filtering events whose values exceed a certain threshold.
- Aggregate algorithms: Used to calculate statistics on data, such as sum, average, count.
- Join Algorithm: Used to combine data from different streams based on one or more keys.
- Windowing algorithm: Used to divide data into time windows and perform calculations on each window.
- Complex event processing algorithm: Used to detect complex events from data streams.

Some algorithms using the operations of Stream Algebra:

- Ford-Fulkerson Algorithm: Classic algorithm for solving maximum stream problem in graphs.
- Edmonds-Karp algorithm: A variant of the Ford-Fulkerson algorithm, more commonly used because of its better time complexity.
- Dinitz Algorithm: Another algorithm for solving the maximum stream problem, which has better time complexity than both Ford-Fulkerson and Edmonds-Karp.
- Tree-based algorithms: Some algorithms use tree structures to represent and process stream data.
- Distributed Algorithms: Algorithms designed to run on parallel/distributed systems.

III. APPLICATIONS OF STREAM ALGEBRA IN BIG DATA ANALYTICS

In the context of big data, Stream Algebra stands out for its role in real-time processing. Instead of waiting to analyze data in batches, Stream Algebra allows for rapid analysis as data is generated. This is important in applications such as:

- Detect financial transaction fraud instantly.
- Analyze user behavior on online platforms to optimize experience.
- Predict and respond to emergency events, such as natural disaster forecasting from sensor data.
- IoT System.
- Monitor sensor data in real time to predict equipment failures.
- Supply Chain Management.
- Track data streams from RFID devices for inventory management.
- …

The paper **[6]**published in the journal Data in July 2024 focuses on improving the performance of Complex Event Processing systems through database indexing strategies. The main content of the paper is to introduce CEP, which is an important processing for real-time analysis of continuous event streams in many fields such as finance, logistics, and security. With the increase in the volume and complexity of event data, optimizing the performance of CEP systems becomes a key challenge. The paper also proposes a new indexing technique called Hierarchical Temporal Indexing (HTI), which is specifically designed to efficiently process complex event queries. HTI takes advantage of the temporal nature of event data and applies a multi-level indexing method to optimize query execution. By combining temporal indexing with spatial and attribute indexing, HTI aims to speed up the retrieval and processing of related events, thereby improving the overall query performance. This study also evaluates the effectiveness of HTI by implementing complex event queries on multiple CEP systems with different indexing strategies. The performance analysis includes measuring query execution time, resource utilization (CPU, memory, etc.), and analyzing the execution plans and query optimization techniques applied by each system. The research results show a significant improvement in execution time compared to traditional indexing strategies, especially in queries that require processing complex event data. In terms of resource utilization, HTI reduces CPU and memory load, making the CEP system more efficient in resource-constrained environments. Applying HTI results in more efficient query execution plans, reduced I/O operations, and increased data retrieval speed indicating that the query is also optimized.

In terms of technology, Apache Flink and Apache Kafka Streams are two powerful frameworks in the field of stream data processing **[7]**. Both are being actively developed with many new research and improvements. Some highlights of each framework are as follows. Apache Flink has low latency and the ability to process both streams and *batch data* through improvements in memory management, task scheduling, and optimization of

data processing operators. In addition, Flink develops more efficient state management mechanisms, including improvements in fault tolerance, state access efficiency, and scalability. Integration with other systems such as Apache Kafka, Apache Hadoop, and cloud storage systems is also provided in Flink, making it easy to use in complex data architectures. For programming to support self-evolving systems, Flink develops APIs and libraries to support more complex use cases, such as complex event processing, machine learning. Apache Kafka Streams is built on top of Apache Kafka, allowing for streaming data processing right in Kafka. Kafka Streams is similar to Flink, Kafka Streams also focuses on improving performance and scalability to handle large volumes of data, stateful processing, and integration with other systems **[8]**.

Another technology is Google Cloud Dataflow **[9]**, which uses concepts from Stream Algebra to model data transformations as Stream Algebra operators, and optimizes the execution of these transformations efficiently based on the general theoretical foundation from Stream Algebra. Stream Algebra is one of the important theoretical foundations of Google Cloud Dataflow, helping Dataflow model, process, and optimize the execution of data processing pipelines.

IV. CONCLUSION

Stream Algebra plays a crucial role in big data processing, especially when data is generated continuously and needs to be analyzed in real time. Research on Stream Algebra in the context of big data analytics still has many open challenges, requiring attention and efforts from the research community. Some of the main challenges are as follows.

1. **Handling Late Data**: In real life, data often does not arrive in perfect time order. Data may be delayed due to network, system failures, or other issues. How to extend Stream Algebra to handle late data naturally and efficiently, defining operations and rules that allow handling late data while still ensuring the correctness of the results.

2. **State management**: Many streaming data analytics applications require maintaining state, such as summing, averaging, or counting the occurrences of an event. Efficiently managing state in a distributed environment is a complex challenge. The issues are how to represent state in Stream Algebra; how to define operations that allow efficient updating and accessing of state; and how to ensure state consistency in a distributed and fault-tolerant environment.

3. **Complex Event Processing**: This is the problem of detecting complex patterns in data streams; such as a sequence of events, events occurring within a certain time period, or events satisfying a complex condition. The challenge in Stream Algebra is how to extend Stream Algebra so that complex patterns can be represented concisely and efficiently; how to define operations that enable the detection of these patterns in real time.

4. **Scalability and performance**: Big data requires the ability to process huge volumes of data at high speed. Ensuring the performance and scalability of Stream Algebra-based systems is a major challenge. The problem is how to optimize the execution of operators in Stream Algebra to achieve high performance; how to distribute data processing work across multiple servers efficiently.

5. **Integration with other data models**: In practice, data is often stored and processed using many different models, such as relational data, NoSQL data, or graph data. Integrating Stream Algebra with these models is a challenge. Specifically, how to define operations that allow interaction with other data models; how to efficiently convert data between these models.

6. **Representation and ease of use**: Stream Algebra can become complex when representing complex data transformations. Methods are needed to simplify the representation and use of Stream Algebra. The challenge in Stream Algebra is how to develop query languages or APIs based on Stream Algebra that are easy to use and understand; how to provide visual tools to help users understand and manipulate data streams.

7. Exactly-once Semantics**:** Ensuring that each event is processed exactly once, even in the presence of system errors, is a key challenge in stream data processing. The challenge is how to integrate error handling and recovery mechanisms into Stream Algebra to ensure "exactly-once" semantics.

V. REFERENCES

[1] Bernhard Möller (1998). Ideal Stream Algebra. *Lecture Notes in Computer Science* , DOI:10.1007/3-540-49254-2_3, In book: Prospects for Hardware Foundations (pp.69-116). Publisher: LNCS 1546, Springer. Accessed date: Dec 18, 2024 (https://opus.bibliothek.uni-augsburg.de/opus4/frontdoor/deliver/index/docId/26230/file/26230.pdf).

[2] MA Helala, FZ Qureshi and KQ Pu (2022) , A Stream Algebra for Performance Optimization of Large Scale Computer Vision Pipelines, *IEEE Transactions on Pattern Analysis and Machine Intelligence* , vol. 44, no. 2, pp. 905-923, 1 Feb. 2022, doi: 10.1109/TPAMI.2020.3015867.
[3] Microsoft Learn Challenge (2024), Introduction to Stream Analytics windowing functions . Published Dec 17, 2024, https://learn.microsoft.com/en-us/azure/stream-analytics/stream-analytics-window-functions . Accessed date: Dec 20, 2024 .
[4] Bill Bejeck (2024). Mastering Stream Processing - Windowing time semantics . P ublished Feb 29, 2024. https://www.linkedin.com/pulse/mastering-stream-processing-windowing-time-semantics-bill-bejeck-fp50e . Accessed date: Dec 20, 2024 .
[5] Timeplus Teams (2024). Complex Event Processing vs Stream Processing. Published Apr 22, 2024. https://www.timeplus.com/post/complex-event-processing-vs-stream-processing . Accessed date: Dec 20, 2024 .
[6] Maryam Abbasi, Marco V. Bernardo, Paulo Váz, José Silva and Pedro Martins (2024). Optimizing Database Performance in Complex Event Processing through Indexing Strategies. *Data Journal* , 9(8), 93; https://doi.org/10.3390/data9080093 . Accessed date: Dec 21, 2024
[7] Luigi Cerrato, (2023), Apache Flink and Kafka Streams: A Comparative Analysis . Published July 25, 2023 https://bitrock.it/blog/technology/apache-flink-and-kafka-stream-a-comparative-analysis.html . Accessed date: Dec 23, 2024 .
[8] Instaclustr (2024). Apache Flink vs Apache Kafka Streams: Comparing Features & Capabilities. Published February 06, 2024. https://www.instaclustr.com/blog/apache-flink-vs-apache-kafka-streams/ . Accessed date Dec 23, 2024 .
[9] Team Zhar (2024). What Is GCP Dataflow. Published Mar 15, 2024. https://www.zuar.com/blog/what-is-gcp-dataflow/. Accessed date Dec 23, 2024 .

ĐẠI SỐ LUỒNG TRONG VIỆC PHÂN TÍCH DỮ LIỆU LỚN

Trần Văn Lăng
Trường Đại học Ngoại ngữ - Tin học TP.HCM

ABSTRACT — Bài báo này trình bày tổng quan về Đại số luồng (*Stream Algebra*), một nhánh nghiên cứu đã có từ lâu sau khi có sự ra đời của Đại số quan hệ (*Relational Algebra*) sử dụng trong các hệ quản trị cơ sở dữ liệu (DBMS). Khi dữ liệu luồng (*Strean Data*) xuất hiện và cần phải được xử lý theo thời gian thực (*real-time processing*), thì vai trò của Đại số luồng đã rõ nét. Nhưng từ khi xuất hiện khái niệm Dữ liệu lớn (*Big Data*), lĩnh vực này đã được sự quan tâm trong những năm gần đây thì vai trò của Đại số luồng càng được thể hiện. Bên cạnh đó, việc phân tích dữ liệu lớn dựa trên tiếp cận của Đại số luồng cũng được bài báo đặt ra, qua đó góp phần vào việc xác định hướng nghiên cứu trong kỷ nguyên của Khoa học Dữ liệu cho các nghiên cứu viên trẻ, những nghiên cứu sinh muốn tìm cho mình một hướng nghiên cứu còn nhiều thách thức. Đặc biệt tập trung phân tích để đưa ra một số vấn đề còn bỏ ngõ trong việc nghiên cứu ứng dụng Đại số luồng. Bài báo cũng trình bày một số Framework sử dụng Đại số luồng để giúp dữ lý dữ liệu luồng hiệu quả giúp nhanh chóng sử dụng trong nghiên cứu triển khai.

Tran Van Lang is an Associate Professor of Information Technology. He graduated with a Bachelor's degree in Applied Mathematics from the University of Natural Sciences, Ho Chi Minh City in 1982, and received a PhD in Mathematics and Physics from the same university in 1995. During his career, he has held many important positions such as Director of the Ho Chi Minh City Institute of Information Technology, Deputy Director of the Institute of Applied Mechanics and Informatics under the Vietnam Academy of Science and Technology (VAST) ; Head of the Faculty of Information Technology, Lac Hong University and Nguyen Tat Thanh University; Chairman of the Science and Training Council of the HCMC University of Foreign Languages - Information Technology. His research interests include Bioinformatics, Cheminformatics, Parallel and Distributed Computing, Scientific Computing and Computational Intelligence. He is proficient in many programming languages such as FORTRAN, C /C++, Java, Python, Perl.

He has also made numerous contributions to the training of IT human resources in Vietnam, especially in the southern region. He currently supervises 7 PhD students to successfully defend their theses on Mathematical Foundation for Informatics and on Computer Science. More than 100 graduate students have received their master's degrees in IT under his supervision. He is also a member of the Steering Committee as General Chair of the Program Committee of The National Conference on Fundamental and Applied IT Research (FAIR). More detailed information about him can be found at https://fair.conf.vn/~lang.