Big data stream processing

Ovidiu-Cristian Marcu and Pascal Bouvry

Abstract This chapter provides students, industry experts, and researchers a high-level and comprehensive overview of the end-to-end architectures of big data stream processing pipelines, designed to keep them informed of the latest advancements and best practices in the field. It explores crucial components such as source ingestion, state and storage mechanisms, stream query processing, and distributed streaming. Streaming architectural pipelines manage continuous data flows from various sources, necessitating efficient ingestion, processing, and storage systems. These systems are vital for applications that require real-time data processing, including fraud detection, IoT monitoring, patient health tracking, and e-commerce personalization. By integrating edge, cloud, and high-performance computing environments, big data stream processing ensures low latency and high throughput. Each section of this chapter delves into the most critical topics in stream processing, referencing recent literature for further exploration and offering insights into ongoing technological developments.

1 Introduction

Stream processing (SP) serves as a fundamental real-time component of Big Data systems, facilitating efficient data management and rapid insight extraction [62]. Its application spans various sectors, including real-time fraud detection, algorithmic trading, IoT monitoring in smart homes and industrial setups, and telecommunications quality assurance. Additionally, SP is critical in health care for real-time patient monitoring and in retail for personalized e-commerce recommendations and

Ovidiu-Cristian Marcu
University of Luxembourg, Luxembourg, e-mail: ovidiu-cristian.marcu@uni.lu

Pascal Bouvry
University of Luxembourg, Luxembourg, e-mail: pascal.bouvry@uni.lu

dynamic pricing. The ability of SP to handle continuous data flows with low latency is essential for contemporary data-centric applications.

Streaming and data streams form a core part of the digital continuum, integrating Edge, Cloud, and High-Performance Computing (HPC) to enable the real-time processing of high-volume data [72]. As per ETP4HPC SRA5 [1], streaming is indispensable for handling extreme data volumes and connects various systems and sources to meet scientific and commercial processing needs. This is increasingly important as modern computing frameworks like Lambda and Kappa architectures integrate streaming with batch processing and database analytics (evolving into lakehouses) to boost the flexibility and analytical capabilities of Big Data infrastructures.

Federated architectural use cases necessitate a comprehensive end-to-end streaming pipeline. In the modern realm of advanced computing, integrating Cloud and HPC resources is essential for managing large-scale projects like Destination Earth (DestinE) and the Square Kilometre Array (SKA). These projects emphasize the need for a federated architecture that efficiently handles vast data volumes through integrated data lakes.

DestinE aims to develop a highly accurate digital twin of the Earth, requiring robust data streaming and movement solutions to support its complex simulations across federated EuroHPC infrastructures. The project focuses on establishing a data lake with a unified metadata schema to ensure seamless data integration and interoperability across various HPC systems. This setup facilitates effective data acquisition and preprocessing, vital for the mobility of code and data within the federated framework.

The SKA project, set to be the world's largest radio telescope, underscores the importance of federated data lakes in managing a continuous stream of 10 terabits per second of raw data, eventually generating 350 petabytes of data products annually. It adopts an in situ and online data processing approach within centralized HPC systems to optimize data flow and minimize the need for extensive data movement. This approach improves data locality by keeping data close to its point of origin while utilizing nearby computing resources, thus supporting extensive data streaming and integration on a global scale.

These examples illustrate the increasing demands on federated environments to support large-scale, data-intensive projects by improving data streaming capabilities, ensuring efficient data integration, and maintaining system scalability and interoperability across geographically dispersed infrastructures.

Over the past three decades, SP systems and languages have undergone considerable evolution (see references). Today, advanced frameworks like Spark Streaming and Flink are fully integrated into cloud-based lakehouse architectures, such as those offered by Confluent and Databricks, and within proprietary systems like Google's BigLake. Meanwhile, on-premises SP alternatives frequently focus on integrating with Kafka to manage partitioned stream ingestion and storage.

Source ingestion, the initial step in SP, involves integrating data from sensors, logs, and other real-time sources into the pipeline. Efficient data collection is supported by techniques such as publish-subscribe models and scalable ingestion systems, including Kafka and KerA. Following ingestion, the focus shifts to state and storage management, which support the real-time requirements of SP. Internal state management is often handled by systems like Apache Flink, while external storage solutions like Pravega offer dynamic partitioning and elasticity. Additionally, techniques for workload-aware state management and the use of persistent memory significantly enhance the efficiency and scalability of SP applications. Managing backpressure and ensuring data consistency are critical in high-throughput environments.

Stream query processing involves applying continuous queries to data streams, supported by SQL-based frameworks and custom query languages. Distributed streaming architectures, leveraging systems such as Kafka, Flink, and Spark Streaming, ensure scalability and fault tolerance. The chapter also reviews scaling strategies, i.e., scale-in, scale-out, and scale-up techniques, as well as benchmarking tools and fault recovery mechanisms. SQL on streams extends traditional SQL capabilities to real-time data flows, enabling complex queries over streaming data, allowing users to apply familiar SQL syntax to process and analyze data in motion, making SP more accessible to developers who might not be specialized in streaming technologies. Frameworks like Apache Flink, Apache Kafka Streams, and Spark Streaming provide robust support for stream SQL, including a wide range of SQL operations like joins, windows, and aggregations.

Each of the sections below highlights key concepts and terms, offering a gateway to more in-depth technical and research-oriented literature. For those interested in further exploration, relevant state-of-the-art references are provided to enhance understanding and knowledge in these specific areas.

2 Background on Ingestion and Storage for Stream Processing

Stream ingestion involves acquiring real-time data streams for immediate processing or storage. This critical initial step in collecting continuously generated data from sources like IoT devices, sensors, and logs, allows for real-time data analysis. **Storage systems** in streaming pipelines are designed to accommodate both real-time and historical data. They must satisfy low-latency access to incoming data streams for real-time analytics and efficiently query batch data for comprehensive insights. This dual requirement underscores the need for advanced storage solutions that can manage high data throughput and support complex queries, often with low latency.

For many applications such as predictive analytics, combining real-time stream data with historical data is essential for making informed decisions. Storage systems must therefore provide mechanisms to seamlessly access and integrate these diverse data types. For a detailed review of specific storage and ingestion systems in support of SP, readers are directed to a comprehensive study [48] that characterizes various systems based on both functional (partitioning, metadata, search support, message routing, backpressure support) and non-functional aspects (high availability, durability, scalability, latency vs. throughput).

2.1 Data Ingestion: Stream Approaches

”The Many Faces of Publish/Subscribe” [23] provides an extensive analysis of publish-subscribe communication paradigms. This communication model works by allowing subscribers to express interest in an event or a pattern of events and receive notifications when these events occur. The system fully decouples the communicating entities in terms of time, space, and synchronization, which enhances flexibility and scalability, while storage plays a critical role in handling asynchronous messages and ensuring data persistence for delayed processing. **Topic-based** systems (e.g., Apache Kafka [22]) allow subscribers to receive messages based on a specific topic, while content-based ones allow subscribers to receive messages based on the content of the messages, requiring more complex filtering mechanisms.

KerA [78], a scalable and unified stream ingestion and storage engine, employs a **dynamic partitioning** strategy to enhance data ingestion for SP. It addresses the limitations of static stream partitioning used by systems like Apache Kafka, which often trades performance for simplicity. KerA dynamically creates stream partitions according to the load, improving ingestion throughput, reducing processing latency, and increasing scalability. The **push-based** [79] integration of KerA with Apache Flink, a modern SP engine, facilitates efficient data flow management from ingestion to processing, in a unified ingestion-processing streaming architecture. Streaming data are actively sent to Flink consumers as soon as they become available in KerA, instead of the conventional **pull-based** method where stream sources repeatedly requests data, potentially over network. A push-based approach will minimize latency and can maximize throughput by eliminating frequent data requests and waiting times, making the streaming pipeline more efficient in processing real-time data streams.

Backpressure is a critical flow control mechanism in high-throughput streaming systems (like Apache Kafka, Apache Flink, and Apache Spark), designed to prevent the processing components from being overwhelmed when the rate of incoming data surpasses the system's processing capacity. Backpressure ensures data is processed at a manageable rate, safeguarding against data loss and system overload. Implementation strategies include rate limiting, buffer management and dynamic rescaling.

2.2 Storage Management in Support of Fast data

Internal state in Apache Flink [5] refers to the stream data managed by processing operators that reflect the history of incoming data streams. This state is essential for functions like aggregations, windows, and joins, which require knowledge of past data to compute results. Flink's management of state for large-scale applications is complex, integrating state directly into the SP workflow, and utilizes a consistent, distributed snapshot mechanism to handle state across its distributed architecture.

This allows Flink to provide strong consistency guarantees (”exactly-once”) and fault tolerance.

Exposing the **internal state** of stream processors for analytics, monitoring, and debugging purposes presents both opportunities and challenges [16]. While this visibility enables advanced operational functionalities, it incurs a performance overhead. The added latency (milliseconds of delay), must be considered when designing systems that require real-time data processing capabilities.

External storage for streaming state. The storage manager for streams (SMS) system [2] is designed to address the rigorous performance requirements of data-intensive SP through a general-purpose, tunable storage management framework. SMS integrates knowledge of data access patterns, such as updates and queries, into its framework, to handle the dynamic nature of SP, which often involves rapid changes in data state and frequent queries.

Workload-aware stream state management [8] involves configuring state backends tailored to the specific state requirements of each SP operator. This approach leverages the fact that many streaming operators are instantiated once and are long-running, allowing their state types, sizes, and access patterns to be either inferred at compile time or learned during execution. The main goal is to optimize state management to reduce latency and increase throughput by eliminating inefficiencies found in general-purpose, one-size-fits-all state management systems (e.g., that rely on LSM-based key-value stores to manage the state of long-running computations).

FlowKV [42] is introduced as a persistent store designed to handle the large-scale state management needs of streaming applications, specifically tailored for SP engines (SPE). Unlike traditional key-value stores, FlowKV utilizes insights from streaming applications, aligning data storage with the semantic requirements of data access within SP. It systematically categorizes **window operations**, which are fundamental to SP, based on their **data access patterns**. These are then optimized with specialized in-memory and on-disk structures to match these patterns, enhancing performance efficiency significantly. FlowKV differentiates itself by employing a semantic-aware approach, where data access patterns dictated by window operations guide the structure and optimization of storage solutions.

Cloud stream storage solutions typically offer elasticity, scalability, and high throughput to handle real-time data processing efficiently. These systems support dynamic scaling of resources to meet varying loads, ensuring that storage capacity and processing power each adjust as needed without manual intervention. This is essential for applications like live video streaming, real-time analytics, and IoT data processing, where data flows can be unpredictable and highly variable. The BigLake [77] system enhances Google BigQuery’s storage for both batch and streaming data, creating a unified multi-cloud lakehouse architecture that supports high-throughput stream ingestion. It provides scalable, real-time data processing capabilities with fine-grained access control, ensuring consistent governance and security across all data formats.

Pravega [24, 25], a novel open-source **elastic stream storage system**, stands out for its auto-scaling capabilities which allow to dynamically adjust data stream parallelism in response to workload changes. Pravega integrates smoothly with Apache

Flink to enable advanced real-time analytics and has built-in transaction support that ensures data integrity. Its scalable architecture is designed to handle high throughput and provide strong consistency guarantees, making it suitable for applications that cannot tolerate data loss or anomalies.

Utilizing HPC hardware for SP is a more recent trend that uses co-design of system and application logic to maximize performance of stateful SP.

”Efficient State Management with Persistent Memory” [26] explores the potential of Persistent Memory (PMem), such as Intel Optane, to significantly enhance state management in data SP systems. A benchmarking framework, PerMA-Bench, is developed to assess the performance of PMem across various configurations, providing insights into its behavior under different conditions. The thesis also introduces Viper, a hybrid PMem-DRAM key-value store optimized for PMem’s unique capabilities, demonstrating performance improvements in data insertion and retrieval operations. Furthermore, it explores the integration of PMem into larger systems like SP engines, showcasing its advantages over traditional setups. The research emphasizes PMem’s role in bridging the gap between volatile memory and persistent storage.

Stateful SP refers to the capability of a SPE to maintain state across multiple input data records. This allows for complex computations where the outcome depends not just on the current record, but also on previously received data. Such state is typically maintained in memory for low-latency access and can be used for various operations like aggregations, windowing, or pattern detection, enhancing the SP engine’s ability to handle complex, real-time analytics.

Remote Direct Memory Access (**RDMA**) enhances mutable stream state sharing in distributed systems by bypassing the CPU, thereby reducing latency and CPU load. Its low-latency, high-throughput capabilities improve state synchronization across nodes, crucial for stateful SP systems. By enabling direct memory access between nodes without CPU involvement, RDMA minimizes synchronization time, maintaining state consistency and supporting real-time performance and scalability in modern SP architectures where timely state updates are essential. The work titled ”Rethinking Stateful Stream Processing with RDMA” [32] discusses the integration of RDMA to significantly enhance the performance of stateful stream processing. The authors introduce ”Slash”, a novel SPE that leverages RDMA to optimize the execution of distributed streaming computations. Unlike traditional SPEs that often rely on costly data re-partitioning and struggle with the real-time constraints of stream processing, Slash utilizes RDMA to efficiently share mutable state across nodes, reducing the latency and overhead associated with state synchronization. Recent advancements in stream processing have led to the development of general-purpose partitioned stateful operators on GPUs, which maintain and update internal state partitioned by key, facilitating scalable data stream management on heterogeneous architectures [41]. WindFlow [40] accelerates pipelines of interconnected operators by efficiently moving and keeping data on the GPU.

2.3 Future Challenges

The evolution of SP systems (SPS) hinges on efficiently integrating many-core CPUs, GPUs, RDMA, persistent memory and advanced networking for real-time stream ingestion, state management and low-latency stateful processing. Effective state management is core to SPS efficiency, particularly through **workload-aware strategies**, dynamically adjusts storage and computation to workload demands, optimizing performance and reducing resource overhead. Distributing and synchronizing state across nodes via **sharding** and **state replication** is essential for consistency and reliability in distributed scalable SPS.

Handling the temporal aspects of data, particularly the **order of streams** and **out-of-order data**, is a fundamental challenge in stream processing. Implementing effective strategies to manage time skew and reordering mechanisms is essential to ensure accurate and timely data analysis, all while leveraging HPC hardware. **Access patterns** such as update, share, read, and query within stream processing require robust implementations to support real-time SP. Techniques like **event time processing**, **windowing functions**, and **stateful operators** will have to be revised.

Overall, rethinking stream processing to leverage HPC hardware will foster more robust, energy-efficient and performant SPS.

3 Execution Architectures for Stream Processing

The architectural design of SPS significantly influences their ability to handle large-scale, real-time data streams effectively on the digital continuum. This section explores various execution architectures that cater to diverse operational demands and scalability requirements in SP. The focus is on understanding different scaling strategies such as scale-in, scale-out, and scale-up, and their implications on system performance and fault tolerance.

Scaling strategies ensure that SPS can adapt to fluctuating data volumes and processing loads. **Scale-in** techniques optimize resource usage within a single node, enhancing its ability to handle large states efficiently. **Scale-out** strategies involve distributing the workload across multiple nodes to improve throughput and resilience, while **scale-up** approaches focus on increasing the capacity of existing hardware to handle more extensive workloads.

3.1 Single-node execution

Scabbard [17] addresses the challenges of implementing fault tolerance in single-node multi-core SPE. Single-node SPEs can process hundreds of millions of tuples per second but traditionally lack built-in fault tolerance, which limits their practical use despite their high performance. The challenge is to achieve **fault tolerance**

with exactly-once semantics without significantly impacting performance, given the constrained I/O bandwidth typical of single-node systems. Scabbard introduces a novel fault-tolerance mechanism that integrates stream and state persistence operations directly into the query workload and the operator dataflow graph. By persisting data after high-selectivity operators that discard irrelevant data, Scabbard reduces the volume of data that needs to be persisted.

Darwin [27] introduces a new paradigm in SP that aims to maximize hardware utilization on single and multi-node systems while supporting large recoverable states and crash recovery. Traditional scale-out systems like Apache Flink and Spark Streaming, while effective in handling large datasets, often lead to resource inefficiencies due to their reliance on extensive clusters. Darwin proposes **scale-in processing** to overcome these inefficiencies by fully utilizing existing hardware without requiring scale-up or extensive scale-out, thereby reducing infrastructure needs. Darwin supports recoverable, larger-than-memory states through advanced storage techniques and integration with modern storage technologies like Persistent Memory (PMem), which bridges the gap between the speed of DRAM and the persistence of SSDs.

Discussion. Single-node SPS suit complex event processing (CEP) when prioritizing low latency and resource-constrained environments, especially with improvements in fault tolerance and processing efficiency. In contrast, scale-out/up architectures are better for handling large data volumes and complex processing needs. They enhance CEP functionalities across distributed nodes, improving resource efficiency and system resilience. The reference [12] underscores this by discussing the integration of CEP with modern execution architectures in SPS, highlighting a shift towards hybrid models that merge CEP with SP.

3.2 Scale-out Stream Processors

Hazelcast Jet [19] is designed as a high-performance, low-latency distributed SPE aimed at managing low-latency processing tasks across scale-out environments. Jet focuses on achieving low-latency processing times, particularly at the 99.99th percentile. Jet leverages Hazelcast's In-Memory Data Grid (IMDG) that serves as a highly available and distributed in-memory storage system, while Jet and the IMDG are designed to maximize data locality by partitioning data in a way that aligns with the distribution of the computation tasks.

The transition from Apache Storm to Twitter Heron [57] was driven by the need to overcome Storm's limitations in high-scale environments like Twitter. Storm grouped multiple SP components (spouts and bolts) within a single JVM process, complicating debugging, especially under variable loads or hardware failures. Heron introduced a **process-per-component** approach, enhancing debugging and resource allocation. It also implemented an advanced backpressure system to manage data flow and stabilize performance under high loads. Additionally, Heron streamlined the management of deployments with better cluster integration tools, reducing operational overhead and improving resource efficiency and scalability through better integration with systems like Mesos and YARN.

Apache Flink [67] and Apache Spark Structured Streaming [68] represent two different approaches to SP, which are characterized primarily by their handling of data - either processing one **record at a time or in mini-batches**. Apache Flink processes data in a true streaming fashion, meaning it processes records individually as they arrive. This approach allows Flink to handle event time processing more naturally and efficiently. Flink's architecture supports full SP capabilities, where each event is handled as it comes, and window operations can be applied flexibly based on the event time, ingestion time, or processing time. This results in lower latency processing because there's no need to wait for a batch to fill up before processing begins. In contrast, Apache Spark Structured Streaming processes data using a micro-batch approach, where incoming streams of data are treated as a series of small batches. This allows Spark to leverage its fast batch processing capabilities to handle streaming data. While this approach simplifies the integration of batch and streaming data models, it inherently introduces higher latency due to the batching interval. However, Spark Structured Streaming also supports event-time aggregation and windowing, which can handle out-of-order data or late data by specifying watermark delays.

3.3 Scale-up Engines

StreamBox [45] introduces several novel features to scale up SP efficiently on a multi-core machine, emphasizing the parallelism and memory hierarchy of modern hardware. StreamBox organizes **out-of-order records** into epochs delineated by watermarks, which define the temporal bounds of the data. It introduces a novel concept of processing epochs out-of-order within each transform, allowing more flexibility and parallelism in handling stream data. StreamBox is designed to steer streams to optimize NUMA (Non-Uniform Memory Access) locality. It organizes pipeline state based on the output window size, placing records that fall within the same windows contiguously in memory.

WindFlow [44] introduces several novel features for optimizing SP on many-core systems, focusing on a unified approach to maximize both throughput and latency. WindFlow uses a formalized software engineering approach based on assembling customizable **building blocks**. These building blocks, which are recurring dataflow compositions of interconnected activities, allow developers to easily express optimizations and tailor the system to specific needs. WindFlow leverages parallel building blocks to optimize the use of many-core architectures. This includes strategies like operator pinning, which assigns specific operations to specific cores, improving performance by taking advantage of the hardware's capabilities.

4 Processing Over Clouds and HPC Data-centers

This section illustrate effective strategies for managing streaming across multi-site clouds, with a particular focus on the challenges and solutions related to data ingestion, transfer, and movement. It aims to provide insights into optimizing data handling within distributed cloud environments and HPC data centers, for enhancing efficiency and scalability in processing large datasets across geographical and architectural boundaries.

4.1 Data Movement

To enhance the capabilities of applications that integrate real-time and batch data, the unified architecture proposed in [6] aims to optimize these processes across multi-site clouds. By merging the functionalities of stream ingestion and file transfer into a single framework, the architecture supports both low-latency operations for small-sized datasets and high-throughput activities for large data volumes. This approach help users by reducing the complexity associated with managing distinct systems for different data types and scales. The SciStream [7] architecture focuses on facilitating fast **memory-to-memory data streaming** between federated scientific instruments, crucial for processing high-throughput data efficiently in real-time. It addresses the limitations in traditional data transfer methods that involve intermediate file systems, which can significantly slow down data processing due to high disk latency. JetStream [3] proposes an innovative approach to streaming across multi-site clouds by leveraging batch-based data transfers, optimizing for both latency and bandwidth utilization. JetStream's core innovation lies in its ability to adapt the size of event batches in real-time, based on cloud conditions and network parameters.

4.2 Data Management Across the Digital Continuum

NebulaStream [31] presents a specialized **IoT data management** system designed to handle the complex requirements of IoT environments. It integrates the diverse and distributed nature of IoT devices into a coherent framework that enables efficient data streaming and management. The article ”IoT Stream Processing and Analytics in the Fog” [55] highlights the advantages of fog computing in improving data SP for IoT applications. **Fog computing** reduces latency and enhances network capacity by processing data closer to its sources, like IoT devices. The survey ”Complex Event Recognition in the Big Data Era” [61] addresses the challenges and methodologies of **complex event recognition** (CER) across geo-distributed environments, emphasizing the management of large-scale, disparate data sources in real time. One major challenge highlighted is communication scalability. Efficient strategies are needed to manage the extensive data transfer between distributed nodes to minimize latency and optimize resource utilization.

5 Performance Benchmarking in Stream Processing

Performance benchmarking in SPS is crucial to understand the capabilities and limitations of various frameworks under different workloads and scenarios. Key performance metrics such as latency, throughput, scalability, and fast recovery can provide a comprehensive picture of system performance and robustness of a SPS. **Latency** measures the time taken for a data point to travel through the SPS from source-entry to sink-exit. Factors affecting latency include network delays, processing time at each stage, and the efficiency of data management within the system. **Throughput** refers to the amount of data processed per unit of time. It is influenced by the system's ability to parallelize tasks, the efficiency of the algorithms used, and the underlying hardware capabilities. **Scalability** is the ability of a system to handle increasing amounts of workload without compromising on performance. It can be measured in terms of system capacity to grow either vertically (scale up) or horizontally (scale out). Key considerations include the system's architecture, data partitioning strategies, and load balancing mechanisms. This benchmark [63] introduces the concept of **sustainable throughput** to measure the long-term performance of stream data processing systems without causing increased latency due to back-pressure. The ability to recover quickly from failures is vital for maintaining high availability and data integrity. Fast **recovery** metrics focus on the time it takes for the system to resume operations at normal performance levels after a disruption. This involves fault tolerance techniques, state management efficiency, and the robustness of data replication strategies.

5.1 Shuffling and Data Distribution

Data shuffling in SP refers to the redistribution of SP state across different processing nodes or tasks. Shuffling helps distribute data more evenly across the cluster, enhancing load balancing and preventing bottlenecks. By redistributing state across different nodes, shuffling can increase the fault tolerance of the system as it ensures that the failure of a single node does not lead to significant data loss. The ”ShuffleBench” [21] benchmark addresses the critical aspects of data shuffling in distributed SPE, emphasizing how shuffling impacts performance and scalability across different systems, specifically with configurations for Apache Flink, Hazelcast, Kafka Streams, and Spark Structured Streaming. ShuffleBench tests frameworks by re-partitioning streams to align with stateful operations.

In [53] authors propose two main scalability metrics for SPS: the Resource Demand Metric and the Load Capacity Metric. These metrics assess how resource demands and processing capabilities change with varying loads and resources, aiming to standardize scalability assessments in SP. Additionally, a Lag Trend Metric measures the backlog of unprocessed messages, supporting real-time processing requirements.

5.2 Fault Fast Recovery

Strategies for achieving fault tolerance include: 1) replication - data or processing tasks are replicated across multiple nodes, ensuring that if one node fails, others can take over without loss of data or processing capability; 2) checkpointing - regularly saving the state of a SP pipeline so that it can be recovered from a known good point, minimizing the amount of work lost due to a failure; 3) rebalancing - dynamically redistributing data and tasks across available nodes to adapt to changes in the cluster, such as node failures. [20] further discusses **fault recovery strategies** and their impact on the performance and scalability of SPE like Kafka Streams, Flink, and Spark Structured Streaming by leveraging ShuffleBench.

Runtime adaptation in SP refers to the ability of a system to adjust its behavior in response to changing workload conditions and system dynamics without human intervention. Self-adaptation mechanisms for dynamic workloads include scalability adjustements, dynamic repartitioning, query and state optimizations, backpressure and resource allocation, and impacts fault-recovery.

5.3 Existing Benchmarks

DSPBench [59] is a comprehensive benchmark suite for evaluating data SPS in distributed environments, featuring 15 applications from diverse domains like finance and social networks. It provides detailed workload characterization for each application, facilitating system comparisons on metrics like memory usage and processing cost.

The evaluation of streaming processing frameworks in [60] evaluates Flink, Kafka Streams, and Spark Streaming to process data streams with different workloads. It compares these frameworks on latency, throughput, and resource consumption across several scenarios including burst data handling and complex event processing. Custom parameter tuning and optimization strategies are applied to each framework to assess performance under varied conditions, offering insights into which framework might be best suited for specific use cases or performance criteria.

Kafka Streams employs a recovery mechanism that leverages Kafka's strong durability and replication features. It uses Kafka's log to manage and recover state, enabling fault recovery across distributed systems. When a node or instance fails, Kafka Streams can recover its state from these logs, ensuring minimal downtime and data loss. This recovery process is inherently linked to Kafka's ability to reassign and rebalance partitions among available instances, which can impact recovery time and system performance [15]. A benchmark study [65] on Kafka topic partitioning proposes a methodology to optimize the number of partitions and brokers, crucial for managing data flows in ML/AI analytics within fog computing environments. The study introduces two heuristics for balancing resource use and performance, evaluated through simulations and real-world Kafka cluster experiments.

The benchmark [66] compares message queuing systems including Kafka, RabbitMQ, RocketMQ, ActiveMQ, and Pulsar, emphasizing throughput, latency, and usage scenarios. The evaluation uses a standardized, reproducible testing framework to ensure fairness. Key insights include each system's strengths and suitable application contexts based on their performance metrics and architectural features.

The study [69] provides a systematic mapping of performance in distributed SPS (DSPS), detailing challenges in achieving high-performance and Quality of Service. It discusses various DSPSs and their performance metrics, highlighting the need for better benchmarks. The paper underscores the diversity in DSPS performance, influenced by execution environments and use cases, and calls for a unified approach to evaluating DSPS performance.

In [71], a **catalog of SP optimizations** are discussed, including a detailed examination of trade-offs and assumptions, aimed at enhancing system optimization and guiding engineering decisions.

6 SQL and Continuous Queries on Streams

This section explores the historical development of SQL in the context of streaming data and highlights the challenges encountered when extending SQL principles to continuous data flows. We discuss various models and frameworks designed to facilitate continuous queries on streaming data, examining their functionalities and contributions to real-time data processing. Additionally, we relate on optimizations to enhance the efficiency and performance of streaming SQL queries, such as incremental query merging and resource sharing optimization.

In 2001, a significant advancement in the field of **continuous queries** over data streams [11] was made by proposing a flexible architecture for defining and evaluating these queries. This work addressed semantic issues, efficiency concerns, and identified open research topics, forming the foundation for the Stanford STREAM project aimed at improving data stream management

In 2002, the topic of **streaming models** to support queries [10] was extensively discussed. This research addressed the fundamental need for a new data processing model, where data arrives in multiple, continuous, rapid, time-varying streams. The paper reviewed past work relevant to data stream systems, explored stream query languages, and highlighted the new requirements and challenges in query processing, providing a comprehensive overview of the field and setting the stage for future developments in data stream management systems.

In the paper ”One SQL to Rule Them All” [14], the authors propose integrating robust streaming capabilities into the SQL standard. This includes the use of time-varying relations to manage both static tables and streaming data. The approach leverages **event time semantics** and introduces minimal keyword extensions to handle the materialization of query results. The proposed enhancements aim to utilize the full breadth of standard SQL semantics for SP while ensuring efficient and effective real-time data analysis .

In [28], the paper discusses advanced techniques for **optimizing streaming queries**. These include methods to improve cache utilization and eliminate redundant computations. In the context of SP, **incremental query merging** [29] is a technique designed to enhance resource sharing and efficiency. This method focuses on identifying and maintaining opportunities for sharing among numerous concurrent stream queries by capturing semantic information to enable merging even when there are syntactic differences between queries.

In [30], the paper presents a novel technique for integrating streams and history tables in a relational database system to support **ad hoc queries**. This approach leverages a specialized ring-buffered relation, enabling efficient query processing that combines ephemeral streams with persistent historical data. In [33], the authors present various **algorithms designed for windowed aggregations and joins** on DSPS.

In [34] authors propose a language named Seraph that supports native streaming features for **property graph query languages**. They formally define Seraph’s semantics by integrating SP with property graphs and time-varying relations, treating time as a first-class citizen. Seraph aims to address the lack of continuous query evaluation capabilities in existing graph query languages and is designed to handle real-time data analysis and management.

In Grizzly [36], an **adaptive query compilation** approach optimizes runtime performance for SP by continuously adapting to changing data characteristics. Through Just-In-Time (JIT) compilation, Grizzly dynamically adjusts query execution, combining various profiling techniques to optimize code for modern hardware.

The paper ”Declarative Languages for Big Streaming Data” [37] provides an extensive **overview of SQL-like languages in modern SPE** such as Apache Kafka, Apache Spark, and Apache Flink. These languages, including KSQL, Flink SQL, and Spark SQL, play a crucial role in facilitating SP by simplifying the development and maintenance of streaming applications. The paper discusses how these systems interpret, execute, and optimize continuous queries, highlighting the balance between expressiveness and simplicity. Additionally, it addresses the open research challenges and future directions in the domain of declarative languages for big streaming data processing.

In [38], the authors develop a comprehensive framework to evaluate **GPU-accelerated stream join** algorithms (SJAs). They explore various configurations and parameters, such as join algorithms, parallelization strategies, and GPU types, to identify performance characteristics. Their findings highlight significant performance variations based on these configurations, providing guidelines for selecting optimal SJAs for different use cases. In [39], the authors present ”Slider,” a hardware-conscious algorithm designed for **sliding window aggregation on GPUs**. Slider optimizes GPU performance by selecting appropriate primitives and kernel configurations based on query parameters to addresses under-utilization issues of GPUs in SP.

FineStream, a SP framework for **integrated CPU-GPU architectures**, eliminates data transmission overhead between CPU and GPU by using a unified memory approach, allowing efficient data processing and fine-grained cooperation between CPUs and GPUs [43]. The framework employs **fine-grained scheduling** to assign query operators to the most suitable processor (CPU or GPU) based on performance characteristics, in contrast to traditional methods that might schedule all operators to a single processor. FineStream demonstrates significant performance improvements, primarily due to its ability to co-run operators and minimize data movement overhead. It dynamically adjusts to varying workloads by profiling performance and redistributing resources, ensuring optimal CPU and GPU utilization and maintaining high throughput and low latency even under fluctuating workloads. FineStream supports SQL-based queries with common relational operators like projection, selection, aggregation, group-by, and join, requiring fine-grained operator-device placement for optimal performance [43].

In the context of querying continuous data streams, significant advancements have been made in developing algorithms capable of real-time processing with limited resources. The foundational approach involves processing data items in a single pass and maintaining concise synopses, such as sketches and samples, to estimate query results efficiently. These methods provide approximate results with probabilistic error bounds, ensuring timely insights without extensive computational overhead [73].

The efforts to standardize SQL on streams [13] have produced insightful preliminary lessons, indicating that more time is required for the community to fully grasp and evaluate the diverse solutions currently available in the market. Despite over two decades of work by the community, the complexity and diversity of streaming SQL solutions necessitate a period of learning and adaptation. The variety in existing solutions underscores the need for a gradual approach to standardization, allowing the community to benefit from the practical experiences and validations of these evolving solutions. Thus, while the goal of standardizing SQL for streaming data remains crucial, it is evident that a deeper engagement with the variety of market solutions (including open-source) is essential for a robust and effective standardization process.

7 Related Work: Surveys

The role of SP in the edge-to-cloud continuum is explored in [4], highlighting the integration of IoT Edge devices with Cloud/HPC systems to balance trade-offs in data analytics and machine learning workflows. A comprehensive **chapter on large-scale SP** is provided in [9], covering major design aspects, programming models, and runtime concerns for scalable (distributed) SPS.

The most recent surveys on the evolution of SP and querying on CPU/GPU architectures are detailed in [46] and [47], respectively. The survey in [46] provides a comprehensive overview of the advancements in SPS, covering fundamental aspects such as **out-of-order data management, state management, fault tolerance, high availability, load management, elasticity, and reconfiguration**. It discusses the evolution from the first generation of SPS, focusing on in-order processing, to the second generation, which embraces out-of-order processing and efficient state management. The survey also highlights future trends and open research problems, emphasizing the increasing integration of SP with cloud services and edge computing.

The survey in [47] focuses on the performance implications and optimizations of SP on **heterogeneous CPU-GPU** environments. It explores the challenges and opportunities presented by utilizing GPUs for SP tasks, such as leveraging their parallel processing capabilities to enhance throughput and reduce latency. The survey delves into various techniques for optimizing query execution on hybrid architectures, including **load balancing, efficient memory management, and specialized algorithms tailored for GPU acceleration**. Additionally, it provides insights into the design considerations for developing SP frameworks that can seamlessly operate across both CPUs and GPUs, ensuring efficient resource utilization and improved performance.

In [35], authors review various **window types** in SP for effective data aggregation. Key types include Tumbling, Sliding, and Session Windows, essential for managing the unbounded nature of data streams by segmenting them into manageable chunks.

To ensure consistent service levels in SPS, **runtime adaptation** [18] strategies such as topology and deployment adaptation, processing adjustments, overload management, and fault tolerance enhancements to maintain service amid variable workloads and conditions are categorized within a comprehensive taxonomy that guides future research to develop robust distributed SPS [18].

[51] presents a comprehensive review on the **self-adaptation** in SPS, focusing on the automation of execution management in SPS. This involves dynamic adaptations like adjusting pipeline operator parallelism and batch sizes in response to fluctuating data streams, ensuring efficiency without human intervention. The study establishes a taxonomy for classifying self-adaptive mechanisms and outlines future research directions in enhancing SP adaptiveness, emphasizing the need for validated, efficient self-adaptive SPS.

The article [52] proposes a comprehensive model for distributed data-intensive systems including SP, integrating key design and implementation choices into a unified framework. [54] offers a thorough examination of **parallelization and elasticity** within SP, detailing adaptable parallelism techniques for handling dynamic data streams of varying intensity. The survey outlines methods to adjust parallelism levels based on fluctuating workloads and resource availability, aiming to guide future improvements in SPS's scalability and responsiveness.

[58] discusses **resource management and scheduling** in DSPS, addressing the challenges of deploying systems that meet quality of service requirements while minimizing resource costs. The survey outlines a comprehensive taxonomy covering

resource provisioning, operator parallelization, and task scheduling. It explores the integration of these strategies within cloud environments under specific service level agreements, highlighting the need for efficient, dynamic resource management strategies to support the variable demands of SP applications.

[64] provides a comprehensive survey on **machine learning for streaming** data, emphasizing incremental learning, online learning, and data stream learning. These approaches are crucial for applications where models must continually update from a constant data flow without multiple passes over the data, addressing the big data characteristics of velocity and volume. The paper highlights the transition of these methods from research to industry applications, noting the necessity for efficient and adaptive learners that can handle real-time data processing challenges. Key challenges in the field are detailed, including handling concept drifts, i.e., changes in the underlying data distribution over time, and the need for robust preprocessing methods in streaming environments. The work serves as a critical review of the current state of the art and outlines significant research opportunities and open challenges in both supervised and unsupervised learning contexts for streaming data.

[75] provides an updated overview of machine learning in the context of data streams, emphasizing the need for ongoing development due to rapidly changing industrial and academic landscapes. Unlike prior surveys, this work takes a critical, contemporary approach, addressing the evolving assumptions and challenges within data stream learning. The authors revisit fundamental definitions in the field, such as supervised data-stream learning, and incorporate modern considerations like **concept drift** and temporal dependence.

[70] surveys SP **languages**, essential for real-time big data applications, exploring the development and features of these languages, focusing on their evolution to handle continuous data flows efficiently.

[74] provides a detailed survey on distributed data SP frameworks, delineating the architecture of SPS which include layers such as ingestion, processing, storage, management, and output. They explore various SP engines and assess their capabilities to handle the unique demands of streaming data, such as low latency and fault tolerance. The survey also outlines a taxonomy and comparative study of major SP engines like Storm, Spark Streaming, Flink, and Kafka Streams, emphasizing their design, functionalities, and application in different industries

[76] survey on **transactional SP** (TSP) discusses the integration of real-time stream management with transactional (ACID) properties. It explores the evolution of TSP systems, emphasizing their ability to handle streaming data with the robustness of database systems. Various TSP requirements and methodologies are examined, highlighting the need for both real-time processing and transactional integrity. The paper reviews different approaches to TSP, including integration challenges and design trade-offs, underscoring the complexity of merging streaming data capabilities with transactional consistency.

The tutorial [49] discusses the evolution and future directions of SPS, including discussions on partitioned state, processing guarantees, and state management strategies, detailing the transition from systems without explicit state management to systems that regard state as a first-class citizen. The survey [50] reviews the state-of-the-art research on optimizing data stream processing systems (DSPSs) by leveraging modern hardware capabilities.

Overall, these works comprehensively cover the eight rules for real-time stream processing [56], such as low-latency data flow (keep data moving), SQL on streams querying, handling data imperfections (delayed, out of order, missing), ensuring predictable outcomes, integrating stored (historical) and streaming data, guaranteeing safety and availability, achieving scalability automatically, and maintaining real-time response, providing a holistic overview of current advancements and future directions in the field.

8 Conclusion

Big data stream processing (BDSP) effectively integrates with modern computing architectures, such as cloud and edge computing, enhancing real-time data processing capabilities to meet growing demands. This integration supports advancements in stream ingestion, state management, and distributed processing, which are crucial for the efficiency and scalability of big data applications. Notably, frameworks like Apache Flink and Pravega have made significant progress in managing both internal and external states, providing robust solutions for handling dynamic data streams. The integration of Persistent Memory and RDMA has significantly improved performance and resource efficiency in BDSP, enhancing the management of large-scale, stateful operations. Additionally, workload-aware state management and elastic scaling help BDSP systems stay resilient and performant under varying loads. Under standardization, SQL on streams represents a significant step towards making real-time data processing more accessible to a broader range of users.

This chapter helps readers gain a comprehensive understanding of the current landscape and emerging trends in BDSP. By exploring the discussions on various frameworks, technologies, and methodologies, practitioners and researchers can make informed decisions about the best approaches and tools for their specific streaming needs. As the field continues to evolve, the collaboration between academia and industry will be crucial in driving further advancements. Future research and development efforts should focus on enhancing the interoperability, fault tolerance, energy efficiency, and security of BDSP systems to meet the ever-growing needs of diverse application domains.

9 Acknowledgment

This work is partially funded by the SnT-LuxProvide partnership on bridging clouds and supercomputers and by the Fonds National de la Recherche Luxembourg (FNR) POLLUX program under the SERENITY Project (ref.C22/IS/17395419).

References

1. Malms, M., Cargemel, L., Suarez, E., Mittenzwey, N., Duranton, M., Sezer, S., Prunty, C., Rossé-Laurent, P., Pérez-Harnandez, M., Marazakis, M., Lonsdale, G., Carpenter, P., Antoniu, G., Narasimharmurthy, S., Brinkman, A., Pleiter, D., Haus, U.-U., Krueger, J., Hoppe, H.-C., Haas, R. (2022). ETP4HPC's SRA 5 - Strategic Research Agenda for High-Performance Computing in Europe - 2022. Zenodo.
https://doi.org/10.5281/zenodo.7347009
2. Irina Botan, Gustavo Alonso, Peter M. Fischer, Donald Kossmann, and Nesime Tatbul. 2009. Flexible and scalable storage management for data-intensive stream processing. In Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology (EDBT '09). Association for Computing Machinery, New York, NY, USA, 934–945.
https://doi.org/10.1145/1516360.1516467
3. Radu Tudoran, Alexandru Costan, Olivier Nano, Ivo Santos, Hakan Soncu, et al.. JetStream: Enabling high throughput live event streaming on multi-site clouds. Future Generation Computer Systems, 2016, 54, doi: ⟨10.1016/j.future.2015.01.016⟩.
https://inria.hal.science/hal-01239124
4. Daniel Rosendo, Alexandru Costan, Patrick Valduriez, and Gabriel Antoniu. 2022. Distributed intelligence on the Edge-to-Cloud Continuum: A systematic literature review. J. Parallel Distrib. Comput. 166, C (Aug 2022), 71–94. https://doi.org/10.1016/j.jpdc.2022.04.004
5. Paris Carbone, Stephan Ewen, Gyula Fóra, Seif Haridi, Stefan Richter, and Kostas Tzoumas. 2017. State management in Apache Flink®: consistent stateful distributed stream processing. Proc. VLDB Endow. 10, 12 (August 2017), 1718–1729. https://doi.org/10.14778/3137765.3137777
6. M. A. Tariq, O. -C. Marcu, G. Danoy and P. Bouvry, "Towards Unified Data Ingestion and Transfer for the Computing Continuum," 2023 IEEE International Conference on Big Data (BigData), Sorrento, Italy, 2023, pp. 1978-1981, doi: 10.1109/BigData59044.2023.10386254.
7. Joaquin Chung, Wojciech Zacherek, AJ Wisniewski, Zhengchun Liu, Tekin Bicer, Rajkumar Kettimuthu, and Ian Foster. 2022. SciStream: Architecture and Toolkit for Data Streaming between Federated Science Instruments. In Proceedings of the 31st International Symposium on High-Performance Parallel and Distributed Computing (HPDC '22). Association for Computing Machinery, New York, NY, USA, 185–198.
https://doi.org/10.1145/3502181.3531475
8. Vasiliki Kalavri, John Liagouris. In support of workload-aware streaming state management. In 12th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 20), USENIX Association, Jul. 2020.
https://www.usenix.org/conference/hotstorage20/presentation/kalavri.
9. Paris Carbone, Gábor E. Gévay, Gábor Hermann, Asterios Katsifodimos, Juan Soto, Volker Markl, and Seif Haridi (2017). Large-Scale Data Stream Processing Systems. In: Zomaya, A., Sakr, S. (eds) Handbook of Big Data Technologies. Springer, Cham.
10. Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. 2002. Models and issues in data stream systems. In Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems (PODS '02). Association for Computing Machinery, New York, NY, USA, 1–16. https://doi.org/10.1145/543613.543615
11. Shivnath Babu and Jennifer Widom. 2001. Continuous queries over data streams. SIGMOD Rec. 30, 3 (September 2001), 109–120. https://doi.org/10.1145/603867.603884
12. Ariane Ziehn, Philipp M. Grulich, Steffen Zeuch, and Volker Markl (EDBT 2024). Bridging the Gap: Complex Event Processing on Stream Processing Systems. Available: http://dx.doi.org/10.48786/edbt.2024.39.
13. Sabina Petride, Dan Sotolongo, Jan Michels, Andrew Witkowski, Cara Haas, and Jim Hughes (2023). "Lessons Learned from Efforts to Standardize Streaming In SQL." *arXiv preprint*, arXiv:2311.03476.

14. Edmon Begoli, Tyler Akidau, Fabian Hueske, Julian Hyde, Kathryn Knight, and Kenneth Knowles. 2019. One SQL to Rule Them All - an Efficient and Syntactically Idiomatic Approach to Management of Streams and Tables. In Proceedings of the 2019 International Conference on Management of Data (SIGMOD '19). Association for Computing Machinery, New York, NY, USA, 1757–1772. https://doi.org/10.1145/3299869.3314040
15. Adriano Vogel, Sören Henning, Esteban Perez-Wohlfeil, Otmar Ertl, and Rick Rabiser (2024). "High-level Stream Processing: A Complementary Analysis of Fault Recovery." *arXiv preprint*, arXiv:2405.07917.
16. J. Verheijde, V. Karakoidas, M. Fragkoulis and A. Katsifodimos, "S-QUERY: Opening the Black Box of Internal Stream Processor State," 2022 IEEE 38th International Conference on Data Engineering (ICDE), Kuala Lumpur, Malaysia, 2022, pp. 1314-1327, doi: 10.1109/ICDE53745.2022.00103.
17. Georgios Theodorakis, Fotios Kounelis, Peter Pietzuch, and Holger Pirk. 2021. Scabbard: single-node fault-tolerant stream processing. Proc. VLDB Endow. 15, 2 (October 2021), 361–374. https://doi.org/10.14778/3489496.3489515
18. Valeria Cardellini, Francesco Lo Presti, Matteo Nardelli, and Gabriele Russo Russo. 2022. Runtime Adaptation of Data Stream Processing Systems: The State of the Art. ACM Comput. Surv. 54, 11s, Article 237 (January 2022), 36 pages. https://doi.org/10.1145/3514496
19. Can Gencer, Marko Topolnik, Viliam Ďurina, Emin Demirci, Ensar B. Kahveci, Ali Gürbüz, Ondřej Lukáš, József Bartók, Grzegorz Gierlach, František Hartman, Ufuk Yılmaz, Mehmet Doğan, Mohamed Mandouh, Marios Fragkoulis, and Asterios Katsifodimos. 2021. Hazelcast jet: low-latency stream processing at the 99.99th percentile. Proc. VLDB Endow. 14, 12 (July 2021), 3110–3121. https://doi.org/10.14778/3476311.3476387
20. Adriano Vogel, Sören Henning, Esteban Perez-Wohlfeil, Otmar Ertl, and Rick Rabiser (2024). "A Comprehensive Benchmarking Analysis of Fault Recovery in Stream Processing Frameworks." *arXiv preprint*, arXiv:2404.06203.
21. Sören Henning, Adriano Vogel, Michael Leichtfried, Otmar Ertl, and Rick Rabiser. 2024. ShuffleBench: A Benchmark for Large-Scale Data Shuffling Operations with Distributed Stream Processing Frameworks. In Proceedings of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE '24). Association for Computing Machinery, New York, NY, USA, 2–13. https://doi.org/10.1145/3629526.3645036
22. T. P. Raptis and A. Passarella, "A Survey on Networked Data Streaming With Apache Kafka," in IEEE Access, vol. 11, pp. 85333-85350, 2023, doi: 10.1109/ACCESS.2023.3303810
23. Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. 2003. The many faces of publish/subscribe. ACM Comput. Surv. 35, 2 (June 2003), 114–131. https://doi.org/10.1145/857076.857078
24. Raúl Gracia-Tinedo, Flavio Junqueira, Brian Zhou, Yimin Xiong, and Luis Liu. 2023. Practical Storage-Compute Elasticity for Stream Data Processing. In Proceedings of the 24th International Middleware Conference: Industrial Track (Middleware '23). Association for Computing Machinery, New York, NY, USA, 1–7. https://doi.org/10.1145/3626562.3626828
25. Raúl Gracia-Tinedo, Flavio Junqueira, Tom Kaitchuck, and Sachin Joshi. 2023. Pravega: A Tiered Storage System for Data Streams. In Proceedings of the 24th International Middleware Conference (Middleware '23). Association for Computing Machinery, New York, NY, USA, 165–177. https://doi.org/10.1145/3590140.3629113
26. Lawrence Benson. Efficient State Management with Persistent Memory. Published online on the Publication Server of the University of Potsdam: https://doi.org/10.25932/publishup-62563 https://nbn-resolving.org/urn:nbn:de:kobv:517-opus4-625637
27. Lawrence Benson, Tilmann Rabl. Darwin: Scale-In Stream Processing. CIDR 2022. https://www.cidrdb.org/cidr2022/papers/p34-benson.pdf
28. Anand Jayarajan, Wei Zhao, Yudi Sun, and Gennady Pekhimenko. 2023. TiLT: A Time-Centric Approach for Stream Query Optimization and Parallelization. In Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS 2023). Association for Computing Machinery, New York, NY, USA, 818–832. https://doi.org/10.1145/3575693.3575704

29. Ankit Chaudhary, Steffen Zeuch, Volker Markl, Jeyhun Karimov. EDBT 2023. Incremental Stream Query Merging. https://openproceedings.org/2023/conf/edbt/3-paper-69.pdf http://dx.doi.org/10.48786/edbt.2023.51
30. Christian Winter, Thomas Neumann and Alfons Kemper. Relation-Based In-Database Stream Processing. Joint Workshops at 49th International Conference on Very Large Data Bases (VLDBW'23) — Second International Workshop on Composable Data Management Systems (CDMS'23), August 28 - September 1, 2023, Vancouver, Canada. https://ceur-ws.org/Vol-3462/CDMS7.pdf
31. Zeuch, S., Chatziliadis, X., Chaudhary, A. et al. NebulaStream: Data Management for the Internet of Things. Datenbank Spektrum 22, 131–141 (2022). https://doi.org/10.1007/s13222-022-00415-0
32. Bonaventura Del Monte, Steffen Zeuch, Tilmann Rabl, and Volker Markl. 2022. Rethinking Stateful Stream Processing with RDMA. In Proceedings of the 2022 International Conference on Management of Data (SIGMOD '22). Association for Computing Machinery, New York, NY, USA, 1078–1092. https://doi.org/10.1145/3514221.3517826
33. Verwiebe, J., Grulich, P.M., Traub, J. et al. Algorithms for Windowed Aggregations and Joins on Distributed Stream Processing Systems. Datenbank Spektrum 22, 99–107 (2022). https://doi.org/10.1007/s13222-022-00417-y
34. Rost, Christopher, Riccardo Tommasini, Angela Bonifati, Emanuele Della Valle, Erhard Rahm, Keith W. Hare, Stefan Plantikow, Petra Selmer and Hannes Voigt. "Seraph: Continuous Queries on Property Graph Streams." International Conference on Extending Database Technology (2024). http://dx.doi.org/10.48786/edbt.2024.21
35. Verwiebe, J., Grulich, P.M., Traub, J. et al. Survey of window types for aggregation in stream processing systems. The VLDB Journal 32, 985–1011 (2023). https://doi.org/10.1007/s00778-022-00778-6
36. Philipp M. Grulich, Breß Sebastian, Steffen Zeuch, Jonas Traub, Janis von Bleichert, Zongxiong Chen, Tilmann Rabl, and Volker Markl. 2020. Grizzly: Efficient Stream Processing Through Adaptive Query Compilation. In Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (SIGMOD '20). Association for Computing Machinery, New York, NY, USA, 2487–2503. https://doi.org/10.1145/3318464.3389739
37. Riccardo Tommasini, Sherif Sakr, Emanuele Della Valle, and Hojjat Jafarpour. EDBT 2020. Declarative Languages for Big Streaming Data. Slides at http://streaminglangs.io/slides/edbt20.pdf. http://dx.doi.org/10.5441/002/edbt.2020.84
38. Dwi P. A. Nugroho, Philipp M. Grulich, Steffen Zeuch, Clemens Lutz, Stefano Bortoli, Volker Markl. Benchmarking Stream Join Algorithms on GPUs: A Framework and its Application to the State-of-the-art. In Letizia Tanca, Qiong Luo 0001, Giuseppe Polese, Loredana Caruccio, Xavier Oriol, Donatella Firmani, editors, Proceedings 27th International Conference on Extending Database Technology, EDBT 2024, Paestum, Italy, March 25 - March 28. pages 188-200, OpenProceedings.org, 2024. https://doi.org/10.48786/edbt.2024.17
39. Georgios Michas, Periklis Chrysogelos, Ioannis Mytilinis, and Anastasia Ailamaki. 2021. Hardware-Conscious Sliding Window Aggregation on GPUs. In Proceedings of the 17th International Workshop on Data Management on New Hardware (DAMON '21). Association for Computing Machinery, New York, NY, USA, Article 13, 1–5. https://doi.org/10.1145/3465998.3466014
40. Gabriele Mencagli, Massimo Torquati, Dalvan Griebler, Alessandra Fais, and Marco Danelutto. 2024. General-purpose data stream processing on heterogeneous architectures with WindFlow. J. Parallel Distrib. Comput. 184, C (Feb 2024). https://doi.org/10.1016/j.jpdc.2023.104782
41. G. Mencagli, D. Griebler and M. Danelutto, "Towards Parallel Data Stream Processing on System-on-Chip CPU+GPU Devices," 2022 30th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP), Valladolid, Spain, 2022, pp. 34-38, doi: 10.1109/PDP55904.2022.00014
42. Gyewon Lee, Jaewoo Maeng, Jinsol Park, Jangho Seo, Haeyoon Cho, Youngseok Yang, Taegeon Um, Jongsung Lee, Jae W. Lee, and Byung-Gon Chun. 2023. FlowKV: A Semantic-Aware

Store for Large-Scale State Management of Stream Processing Engines. In Proceedings of the Eighteenth European Conference on Computer Systems (EuroSys '23). Association for Computing Machinery, New York, NY, USA, 768–783. https://doi.org/10.1145/3552326.3567493
43. F. Zhang, Chenyang Zhang, Lin Yang, Shuhao Zhang, Bingsheng He, Wei Lu , and Xiaoyong Du. "Fine-Grained Multi-Query Stream Processing on Integrated Architectures," in IEEE Transactions on Parallel and Distributed Systems, vol. 32, no. 9, pp. 2303-2320, 1 Sept. 2021, doi: 10.1109/TPDS.2021.3066407
44. G. Mencagli, M. Torquati, A. Cardaci, A. Fais, L. Rinaldi and M. Danelutto, "WindFlow: High-Speed Continuous Stream Processing With Parallel Building Blocks," in IEEE Transactions on Parallel and Distributed Systems, vol. 32, no. 11, pp. 2748-2763, 1 Nov. 2021, doi: 10.1109/TPDS.2021.3073970
45. Hongyu Miao, Heejin Park, Myeongjae Jeon, Gennady Pekhimenko, Kathryn S. McKinley, and Felix Xiaozhu Lin. StreamBox: Modern Stream Processing on a Multicore Machine. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, pages 617–629, Santa Clara, CA, July 2017. USENIX Association. ISBN 978-1-931971-38-6. URL: https://www.usenix.org/conference/atc17/technical-sessions/presentation/miao.
46. Fragkoulis, M., Carbone, P., Kalavri, V. et al. A survey on the evolution of stream processing systems. The VLDB Journal 33, 507–541 (2024). https://doi.org/10.1007/s00778-023-00819-8
47. Viktor Rosenfeld, Sebastian Breß, and Volker Markl. 2022. Query Processing on Heterogeneous CPU/GPU Systems. ACM Comput. Surv. 55, 1, Article 11 (January 2023), 38 pages. https://doi.org/10.1145/3485126
48. Ovidiu-Cristian Marcu, Alexandru Costan, Gabriel Antoniu, María S Pérez-Hernández, Radu Tudoran, and Bogdan Nicolae. Storage and Ingestion Systems in Support of Stream Processing: A Survey. [Technical Report] RT-0501, INRIA Rennes - Bretagne Atlantique and University of Rennes 1, France. 2018, pp.1-33. hal-01939280v2
49. Paris Carbone, Marios Fragkoulis, Vasiliki Kalavri, and Asterios Katsifodimos. 2020. Beyond Analytics: The Evolution of Stream Processing Systems. In Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (SIGMOD '20). Association for Computing Machinery, New York, NY, USA, 2651–2658. https://doi.org/10.1145/3318464.3383131
50. Shuhao Zhang, Feng Zhang, Yingjun Wu, Bingsheng He, and Paul Johns. 2020. Hardware-Conscious Stream Processing: A Survey. SIGMOD Rec. 48, 4 (December 2019), 18–29. https://doi.org/10.1145/3385658.3385662
51. Adriano Vogel, Dalvan Griebler, Marco Danelutto, and Luiz Gustavo Fernandes. Self-adaptation on parallel stream processing: A systematic review. *Concurrency and Computation: Practice and Experience*, 2021. First published: 07 December 2021. DOI: https://doi.org/10.1002/cpe.675910.1002/cpe.6759.
52. Alessandro Margara, Gianpaolo Cugola, Nicolò Felicioni, and Stefano Cilloni. 2023. A Model and Survey of Distributed Data-Intensive Systems. ACM Comput. Surv. 56, 1, Article 16 (January 2024), 69 pages. https://doi.org/10.1145/3604801
53. Sören Henning and Wilhelm Hasselbring. 2021. How to Measure Scalability of Distributed Stream Processing Engines? In Companion of the ACM/SPEC International Conference on Performance Engineering (ICPE '21). Association for Computing Machinery, New York, NY, USA, 85–88. https://doi.org/10.1145/3447545.3451190
54. Henriette Röger and Ruben Mayer. 2019. A Comprehensive Survey on Parallelization and Elasticity in Stream Processing. ACM Comput. Surv. 52, 2, Article 36 (March 2020), 37 pages. https://doi.org/10.1145/3303849
55. S. Yang, "IoT Stream Processing and Analytics in the Fog," in IEEE Communications Magazine, vol. 55, no. 8, pp. 21-27, Aug. 2017, doi: 10.1109/MCOM.2017.1600840.
56. Michael Stonebraker, Uğur Çetintemel, and Stan Zdonik. 2005. The 8 requirements of real-time stream processing. SIGMOD Rec. 34, 4 (December 2005), 42–47. https://doi.org/10.1145/1107499.1107504
57. Sanjeev Kulkarni, Nikunj Bhagat, Maosong Fu, Vikas Kedigehalli, Christopher Kellogg, Sailesh Mittal, Jignesh M. Patel, Karthik Ramasamy, and Siddarth Taneja. 2015. Twitter

Heron: Stream Processing at Scale. In Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data (SIGMOD '15). Association for Computing Machinery, New York, NY, USA, 239–250. https://doi.org/10.1145/2723372.2742788

58. Xunyun Liu and Rajkumar Buyya. 2020. Resource Management and Scheduling in Distributed Stream Processing Systems: A Taxonomy, Review, and Future Directions. ACM Comput. Surv. 53, 3, Article 50 (April 2020), 41 pages. https://doi.org/10.1145/3355399
59. M. V. Bordin, D. Griebler, G. Mencagli, C. F. R. Geyer and L. G. L. Fernandes, "DSPBench: A Suite of Benchmark Applications for Distributed Data Stream Processing Systems," in IEEE Access, vol. 8, pp. 222900-222917, 2020, doi: 10.1109/ACCESS.2020.3043948
60. G. van Dongen and D. Van den Poel, "Evaluation of Stream Processing Frameworks," in IEEE Transactions on Parallel and Distributed Systems, vol. 31, no. 8, pp. 1845-1858, 1 Aug. 2020, doi: 10.1109/TPDS.2020.2978480
61. Giatrakos, N., Alevizos, E., Artikis, A. et al. Complex event recognition in the Big Data era: a survey. The VLDB Journal 29, 313–352 (2020). https://doi.org/10.1007/s00778-019-00557-w
62. Ali Davoudian and Mengchi Liu. 2020. Big Data Systems: A Software Engineering Perspective. ACM Comput. Surv. 53, 5, Article 110 (September 2020), 39 pages. https://doi.org/10.1145/3408314
63. J. Karimov, T. Rabl, A. Katsifodimos, R. Samarev, H. Heiskanen and V. Markl, "Benchmarking Distributed Stream Data Processing Systems," 2018 IEEE 34th International Conference on Data Engineering (ICDE), Paris, France, 2018, pp. 1507-1518, doi: 10.1109/ICDE.2018.00169
64. Heitor Murilo Gomes, Jesse Read, Albert Bifet, Jean Paul Barddal, and João Gama. 2019. Machine learning for streaming data: state of the art, challenges, and opportunities. SIGKDD Explor. Newsl. 21, 2 (December 2019), 6–22. https://doi.org/10.1145/3373464.3373470
65. Theofanis P. Raptis, Claudio Cicconetti, and Andrea Passarella. Efficient topic partitioning of Apache Kafka for high-reliability real-time data streaming applications. *Future Generation Computer Systems*, 2024. Institute of Informatics and Telematics, National Research Council, Pisa, Italy. Received 19 September 2023, Revised 16 November 2023, Accepted 23 December 2023, Available online 8 January 2024, Version of Record 11 January 2024. DOI: https://doi.org/10.1016/j.future.2023.12.02810.1016/j.future.2023.12.028.
66. G. Fu, Y. Zhang and G. Yu, "A Fair Comparison of Message Queuing Systems," in IEEE Access, vol. 9, pp. 421-432, 2021, doi: 10.1109/ACCESS.2020.3046503
67. Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. Apache Flink™: Stream and Batch Processing in a Single Engine. *IEEE Data Engineering Bulletin*, 2015. http://sites.computer.org/debull/A15dec/p28.pdf.
68. Michael Armbrust, Tathagata Das, Joseph Torres, Burak Yavuz, Shixiong Zhu, Reynold Xin, Ali Ghodsi, Ion Stoica, and Matei Zaharia. 2018. Structured Streaming: A Declarative API for Real-Time Applications in Apache Spark. In Proceedings of the 2018 International Conference on Management of Data (SIGMOD '18). Association for Computing Machinery, New York, NY, USA, 601–613. https://doi.org/10.1145/3183713.3190664
69. A. Vogel, S. Henning, O. Ertl and R. Rabiser, "A systematic mapping of performance in distributed stream processing systems," 2023 49th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Durres, Albania, 2023, pp. 293-300, doi: 10.1109/SEAA60479.2023.00052
70. Martin Hirzel, Guillaume Baudart, Angela Bonifati, Emanuele Della Valle, Sherif Sakr, and Akrivi Akrivi Vlachou. 2018. Stream Processing Languages in the Big Data Era. SIGMOD Rec. 47, 2 (June 2018), 29–40. https://doi.org/10.1145/3299887.3299892
71. Martin Hirzel, Robert Soul ́e, Scott Schneider, Bu˘gra Gedik, and Robert Grimm. 2014. A catalog of stream processing optimizations. ACM Comput. Surv. 46, 4, Article 46 (March 2014), 34 pages. DOI: http://dx.doi.org/10.1145/2528412
72. Stephens, R. A survey of stream processing. Acta Informatica 34, 491–541 (1997). https://doi.org/10.1007/s002360050095
73. Garofalakis, M., Gehrke, J., Rastogi, R. (2016). Data Stream Management: A Brave New World. In: Garofalakis, M., Gehrke, J., Rastogi, R. (eds) Data Stream Management. Data-Centric Systems and Applications. Springer, Berlin, Heidelberg.

74. H. Isah, T. Abughofa, S. Mahfuz, D. Ajerla, F. Zulkernine and S. Khan, "A Survey of Distributed Data Stream Processing Frameworks," in IEEE Access, vol. 7, pp. 154300-154316, 2019, doi: 10.1109/ACCESS.2019.2946884
75. Jesse Read and Indrė Žliobaitė. Learning from Data Streams: An Overview and Update. 2023. arXiv:2212.14720 [cs.LG].
76. Zhang, S., Soto, J. Markl, V. A survey on transactional stream processing. The VLDB Journal 33, 451–479 (2024).
77. Justin Levandoski, Garrett Casto, Mingge Deng, Rushabh Desai, Pavan Edara, Thibaud Hottelier, Amir Hormati, Anoop Johnson, Jeff Johnson, Dawid Kurzyniec, Sam McVeety, Prem Ramanathan, Gaurav Saxena, Vidya Shanmugan, and Yuri Volobuev. 2024. BigLake: BigQuery's Evolution toward a Multi-Cloud Lakehouse. In Companion of the 2024 International Conference on Management of Data (SIGMOD/PODS '24). Association for Computing Machinery, New York, NY, USA, 334–346. https://doi.org/10.1145/3626246.3653388
78. Ovidiu-Cristian Marcu, Alexandru Costan, Gabriel Antoniu, María Pérez-Hernández, Bogdan Nicolae, Radu Tudoran, Stefano Bortoli, "KerA: Scalable Data Ingestion for Stream Processing," 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS), Vienna, Austria, 2018, pp. 1480-1485, doi: 10.1109/ICDCS.2018.00152.
79. Marcu, OC., Bouvry, P. (2023). In Support of Push-Based Streaming for the Computing Continuum. In: Nguyen, N.T., et al. Intelligent Information and Database Systems. ACIIDS 2023. Lecture Notes in Computer Science(), vol 13996. Springer, Singapore. https://doi.org/10.1007/978-981-99-5837-$5_2$8