A Survey of Stream Processing System Benchmarks

Wang Yue, Martin Boissier, and Tilmann Rabl

Hasso Platner Institut, Potsdam, Germany
{wang.yue, martin.boissier, tilmann.rabl}@hpi.de

Abstract. Stream processing systems are a fundamental component of modern data processing, enabling timely and efficient handling of streaming data. To assess and compare the capabilities of stream processing systems, various benchmarks have been proposed over the past years. Examples span a wide range of use cases, ranging from benchmarks for enterprise computing to social network analyses and IoT networks. These benchmarks are designed with different focuses and exhibit different characteristics during execution. In this paper, we review existing stream processing benchmarks and analyze them across five dimensions: benchmark type, included workloads, data ingestion, supported systems under test (SUT), and tracked metrics. We compare their similarities and differences, providing a comprehensive overview of existing benchmarks. Finally, we discuss aspects that have been overlooked and highlight those that should be addressed when benchmarking future generations of streaming systems.

1 Introduction

Modern data-intensive applications – such as fraud detection, network monitoring, or Internet of Things (IoT) analytics – produce huge amounts of data. To manage these large-volume and high-speed data streams, specialized stream processing systems (SPSs) have emerged, including Apache Spark Streaming [74], Apache Flink [14], Apache Storm [63], and Kafka Streaming [43, 57]. As the complexity and variety of SPSs continue to grow, the need to evaluate and compare the performance of different SPSs also increases [22]. Benchmarks play an important role in this context, offering different workloads and metrics to evaluate key features of current SPSs, e.g., latency, throughput, fault tolerance, and scalability. These benchmarks are essential for both industry and research, as they offer a reproducible way to assess the performance of an SPS.

The state-of-the-art benchmarks targeting SPSs are rich and diverse in terms of workloads, metrics, and terminologies. This makes it challenging to compare current benchmarks directly and understand the trade-offs involved in these SPSs. Therefore, a comprehensive survey of existing stream processing system benchmarks is necessary.

In this paper, we aim to provide an in-depth overview of current SPS benchmarks and investigate their capabilities. We review previous work and examine

27 benchmark efforts. For each benchmark, we introduce its characteristics, including its designed purpose, data ingestion method, workload pipeline, and conducted experiments, to ease comparisons. Additionally, we propose a classification for current stream processing system benchmarks based on five dimensions. We present a comparative analysis of these benchmarks and highlight their strengths and weaknesses. Furthermore, we discuss the challenges and limitations inherent in current stream processing system benchmarks and offer insights into future trends.

The rest of this paper is organized as follows. In Section 2, we present our classification and analysis of current stream processing system benchmarks. In Section 3, we introduce the benchmarks we collected and discuss their characteristics. In Section 4, we review related work and compare it with ours. In Section 5, we conclude our work and discuss potential future stream processing system benchmarks.

2 Classification

In this section, we propose five dimensions, including type, workloads, data ingestion, system under test, and metrics to classify and analyze benchmarks we present in Section 3. To provide a chronological overview of all publications we have surveyed, Table 1 lists all benchmarks along with their earliest publication and sorts them by year published.

2.1 Type

Type (see Table 1) refers to the purpose for which the authors present the benchmark. We classify benchmarks into four types: benchmark specification (S), benchmark implementation (I), benchmark evaluation (E), and benchmark tools (T). The benchmark specification (S) provides a standardized method for evaluating the performance of systems, similar to the TPC-C and TPC-H benchmarks for relational database systems. Examples in stream processing include Linear Road [8] and YSB [16]. They can be extended to benchmark additional systems, metrics, and workloads, and are also used by other benchmark efforts [45]. The benchmark implementation (I) offers a usable implementation to assess the different systems in terms of performance, scalability, durability, and different workloads. The benchmark evaluation (E) conducts evaluations to analyze the performance of systems with various features and scenarios. For example, Karimov et al. [41] focus on accurately measuring the latency and throughput of systems, while Zeuch et al. [75] explore efficiently utilizing modern hardware. Additionally, benchmark tools (T) aim to generate benchmarks for evaluation purposes, rather than evaluating systems directly.

Table 1: Overview of the type and workloads of existing stream processing system benchmark efforts.

Benchmark	**Year**	**Type**	**Use Case**	**#App**	**#Task**
Theodolite [27]	2021	S	IoT Application	4	7
ESPBench [33]	2021	S	Manufacturing		5
YSB [16]	2016	S	Advertisement Analytic	1	
BigBench [24]	2013	S	E-commerce		30
HiBench [34]	2010	S	Real & Synthetic		29
Linear Road [8]	2004	S	Toll System	3	
Henning et al. [29]	2024	I	Cloud Application	4	7
Vikash et al. [66]	2020	I	IoT Application	1	
DSPBench [12]	2020	I	Real & Synthetic	15	
OSPBench [19]	2020	I	Micro Benchmark		1
Shahverdi et al. [59]	2019	I	Advertisement Analytic	1	
Inoubli et al. [36]	2018	I	Social Media	1	
Truong et al. [65]	2018	I	Micro Benchmark		1
Yang et al. [71]	2018	I	Real Application	2	
RIoTBench [60]	2017	I	IoT Application	5	27
Čermák et al. [15]	2016	I	Network Flow Analysis	1	6
Lopez et al. [45]	2016	I	Threat Detection	1	
Wang et al. [69]	2016	I	Micro Benchmark		3
StreamBench [46]	2014	I	Micro Benchmark		7
ShuffleBench [31]	2024	E	Micro Benchmark	1	
Pfandzelter et al. [51]	2022	E	FaaS Application	2	
Chu et al. [17]	2020	E	Micro Benchmark		4
Zeuch et al. [75]	2019	E	Real Application	3	
Hesse et al. [32]	2018	E	Micro Benchmark		4
Karimov et al. [41]	2018	E	Online Game	1	
SPBench [23]	2023	T	Real Application	4	
Pagliari et al. [50]	2019	T			

2.2 Workloads

We use task count (#Task), application count (#App), and use case to describe workloads in benchmarks (see Table 1). The workloads are defined differently in different benchmark efforts. A workload may consist of one or many tasks, which are also referred to as queries in some benchmark efforts. These typically involve calculations such as aggregation or filtering and can be mapped to operators in the stream processing implementation. We use task count to indicate the total number of tasks performed. Additionally, workloads correspond to applications inspired by use cases. These applications involve multiple procedures, such as data ingestion from Kafka, data parsing and joining, and window aggregation [19]. We count the number of applications using an application count. The use case refers to the scenarios in which the workloads are being applied. For example, Čermák et al. [15] evaluate SPSs based on the network data analysis application. Some benchmarks are not from applications but focus on assessing certain performance characteristics of SPSs, we refer to them as micro benchmarks.

Table 2: Overview of data sets, data ingestion, system under test (SUT), and metrics of existing stream processing system benchmark efforts.

Benchmark	**Dataset**		**Ingestion**				**SUT**							**Metric**						
	Synthetic	Real	Kafka	Generator	Local File	Others	Flink	Storm	Spark Streaming	Kafka Streaming	Samza	Hazelcast	Others	Throughput	Latency	CPU	Memory	Network	Disk I/O	Others
Theodolite [27]	●	○	●	○	○	○	●	○	○	●	○	○	○	○	○	○	○	○	○	●
ESPBench [33]	○	●	●	○	○	○	●	○	●	○	○	●	○	○	●	●	○	○	●	○
YSB [16]	●	○	●	○	○	○	●	●	●	○	○	○	○	●	●	●	○	●	○	○
BigBench [24]	●	○	●	●	○	○	○	○	●	○	○	○	○	○	●	○	○	○	○	○
HiBench [34]	●	○	●	○	○	○	●	●	●	○	○	○	○	●	●	●	●	●	○	○
Linear Road [8]	●	○	○	○	●	○	○	○	○	○	○	○	●[a]	●	●	○	○	○	○	●
Henning et al. [29]	●	○	●	○	○	○	●	○	○	●	●	●	○	○	○	○	○	○	○	●
Vikash et al. [66]	○	●	○	○	○	●[b]	●	●	●	○	○	○	●[c]	●	●	○	○	○	○	●
DSPBench [12]	●	●	●	○	○	○	○	●	●	○	○	○	○	●	●	●	●	●	○	○
OSPBench [19]	○	●	●	○	○	○	●	○	●	●	○	○	●[d]	●	●	●	●	●	●	●
Shahverdi et al. [59]	●	○	●	○	○	○	●	●	●	●	○	●	○	○	●	●	●	○	○	○
Inoubli et al. [36]	○	●	●	○	○	○	●	●	●	○	●	○	○	○	○	●	●	●	●	○
Truong et al. [65]	●	○	○	○	○	●[e]	○	○	○	○	○	○	●[f]	●	●	○	○	○	○	●
Yang et al. [71]	●	●	●	○	○	○	●	●	●	○	○	○	○	○	○	●	○	○	○	○
RIoTBench [60]	○	●	○	○	●	○	●	●	●	○	○	○	○	●	●	●	●	○	○	●

(*continued on next page*)

Čermák et al. [15]	○●	●○○○	○●●○●○○	●○○○○○○
Lopez et al. [45]	○●	●○○○	●●●○○○○	●○○○○○●
Wang et al. [69]	●○	●○○○	●●●○○○○	●●○○○○○
StreamBench [46]	○●	●○○○	○●●○●○○	●●○○○○●
ShuffleBench [31]	●○	●○○○	●○○●○●●[d]	●●○○○○○
Pfandzelter et al. [51]	●○	●○○○	●○○○○○○	○○○○○○●
Chu et al. [17]	●●	●○○○	●●○○○○●[f]	●○○○○○●
Zeuch et al. [75]	●●	●○○○	●●●○○○●[g]	●●○○○○●
Hesse et al. [32]	○●	●○○○	●○●○○○●[h]	○○○○○○●
Karimov et al. [41]	●○	○●○○	●●●○○○○	●●●○●○○
SPBench [23]	○○	○○○○	○○○○○○○	●●●●○○○
Pagliari et al. [50]	●○	○○○○	●○○○○○○	○○○○○○○

[a] Supports systems under test *System X* and *Aurora*.
[b] System ingests data via a *RESTful API* and *MQTT*.
[c] Supports systems under test *Apex* and *Nifi*.
[d] Supports system under test *Structured Streaming*.
[e] Systems under test directly ingests data.
[f] Supports system under test *Heron*.
[g] Supports systems under test *Streambox*, *Saber*, `C++`, and *Java*.
[h] Supports system under test *Apex*.

2.3 Data Ingestion

We define data ingestion to describe how a benchmark inputs data, including dataset and ingestion (see Table 2). The dataset can be categorized into *real* and *synthetic*. *Real* refers to real-world datasets, while *synthetic* indicates data produced by data generators or retrieved from synthetic datasets. Ingestion refers to how benchmarks feed data to evaluate SPSs. Most benchmarks input data from Kafka [5] or directly from data generators. Truong et al. [65] implement data generators within the evaluated systems, Vikash et al. [66] utilize interfaces such as RESTful and MQTT to read data from networks, and RIoTBench [60] and Linear Road [8] read data from local files.

2.4 System Under Test

In Table 2, the system under test (SUT) refers to the evaluated systems, including Apache Flink [14], Apache Storm [63], Apache Spark Streaming [74], Apache Spark Structured [7], Apache Kafka Streaming [43, 57], Apache Apex [3], Apache

Samza [49], Apache NiFi [6], Hazelcast [26], and Heron [44]. Linear Road [8] compares System X (a commercial database) and Aurora [1] (a research prototype). Zeuch et al. [75] evaluate two research prototypes (Streambox[47] and Saber [42]) and two hard-coded programs, written in `C++` and `Java`.

2.5 Metrics

Metrics are used to measure the performance of a system. We list all metrics in Table 2, including throughput, latency, CPU usage, memory usage, network usage, and disk I/O. Additionally, we use others to refer to metrics defined for specific scenarios, such as execution time [17, 32, 75], throughput and latency penalty factor [46], garbage collection [21], and message loss rate [45].

3 Literature Review

We review previous work, selecting the most recognized and widely adopted efforts, including six benchmark specifications, 13 benchmark implementations, six benchmark evaluations, and two benchmark tools. Some benchmark efforts do not have names. To save space and improve readability, we use a few keywords from their titles instead of the full titles for section headings.

3.1 Benchmark Specifications

Theodolite Theodolite [27] aims to create specification-based benchmarks and evaluates the scalability of SPSs, including Flink [28] and Kafka Streaming. It is capable of assessing both the horizontal and vertical scalability of cloud-native applications. It is implemented using microservices and ingests data streams from Kafka. Theodolite proposes four use cases inspired by the Titan Control Center [30], a microservice-based analytics platform for IoT data. Each use case requires a different data input format, so Theodolite designs specific generators for every use case. These generators produce synthetic data.

There are four use cases (applications) with seven tasks. (i) Database storage: Data is directly sent to the database. (ii) Hierarchical aggregation: Data is grouped hierarchically and then aggregated. (iii) Downsampling: Data is processed to decrease its volume. (iv) Aggregation based on time attributes: Data is grouped according to time and then aggregated. Theodolite introduces three new metrics. One measures the number of instances required. The other two indicate whether the instances are sufficient for processing tasks or if there is backpressure.

ESPBench ESPBench [33] is designed for evaluating data stream processing systems within enterprise contexts, where streaming data is combined with structured business data. It includes an example implementation using Apache Beam [4], which is an abstraction layer for defining data processing applications. ESPBench evaluates Hazelcast Jet, Flink, and Spark Streaming, covering core functionalities. The benchmark uses two datasets, one is a real-world sensor dataset, and the other is generated using the TPC-C setting. The message broker is Kafka. Additionally, ESPBench introduces a validator tool to ensure the credibility of query results. This tool calculates the query results and compares them to the evaluated systems.

The systems are evaluated with five queries: (i) Check sensor status: This query calculates the average, minimum, maximum, and overall number of sensor values in tumbling windows of one second. (ii) Determine outliers: Utilizing the Stochastic Outlier Selection algorithm [40] to identify outliers. (iii) Identify errors: Filtering sensor records based on a threshold value. (iv) Check machine power: This query verifies if the power of a machine is unexpectedly low from the sensor data stream and if there is no planned downtime from the business data. (v) Persist processing times: This query measures the processing time and updates the database. ESPBench assesses latency and system load for stream processing systems. System load provides an overview of the CPU and I/O utilization of a server.

Yahoo Streaming Benchmarks Yahoo Streaming Benchmarks (YSB) [16, 70] is an open-source benchmark for three stream processing platforms: Flink, Spark Streaming, and Storm. The benchmark simulates an advertisement analytics pipeline, where multiple advertising campaigns generate JSON events. It is also widely used by other benchmark efforts [17, 60, 71]. To generate the desired input load, the benchmark can run multiple Kafka producer instances. There is only one application that reads JSON events from Kafka and filters out irrelevant events. It takes a windowed count of events per campaign and stores each window in Redis. Windows are aggregated to output results. The benchmark evaluates latency and throughput for three SPSs.

Additionally, based on YSB, Karakaya et al. [45] evaluate Spark Streaming, Flink, and Storm, focusing on throughput, network usage, and CPU usage. There are three benchmark experiments: (i) Throughput measurement: It measures the throughput of systems with varying cluster sizes, parallelism settings, and Kafka partition configurations. (ii) System scalability. (iii) Resource consumption (e.g., network and CPU usage).

BigBench BigBench [24] is an end-to-end benchmark specification based on TPC-DS [52]. It was initially implemented on the Teradata Aster DBMS and then extended to stream processing systems. BigBench enriches TPC-DS's data model to incorporate structured, semi-structured, and unstructured data. BigBench proposes a generic and parallel data generator called Parallel Data Generation Framework (PDGF, [55]) that can handle the volume, variety, and velocity aspects of big data systems. PDGF can generate large amounts of data based on a scale factor. The BigBench consists of 30 queries, which are designed to cover one business dimension and three technical dimensions. From a business perspective, the queries are based on nine big data retail levers identified from a McKinsey report on big data in the retail industry. From a technical perspective, the queries cover three dimensions: data sources, processing types, and analytical techniques. Data sources contain structured, semi-structured, and unstructured data, and processing types have declarative processing (SQL-like queries) and procedural processing (MapReduce). Analytical techniques include statistical analysis, data mining, and simple reporting.

Rabl et al. [54] discuss the vision for BigBench, emphasizing several key enhancements. These include the integration of graph analytics, machine learning tasks, multimedia analysis, stream processing, and key-value processing modules. Furthermore, Ivanov et al. [39] present an extension of the BigBench benchmark to incorporate Spark Streaming. It executes five queries periodically on a data stream. Also, they measure the latency of the SPSs.

HiBench HiBench [34, 37, 72] is an open-source benchmark for Hadoop that includes both synthetic micro benchmarks and real-world applications. Initially proposed by Intel in 2010, HiBench originally focused on evaluating MapReduce [34]. It evaluates various factors such as job running time, throughput, HDFS bandwidth, system resource utilization, and data access patterns. In its latest version (7.1.1), HiBench has been updated to support stream processing systems like Flink, Storm, and Spark Streaming [37]. HiBench includes its own data generators and employs Kafka to send data to the test cluster. The metrics used for evaluations are throughput, latency, CPU usage, memory usage, and network usage.

HiBench contains 29 tasks divided into six categories. The first five categories are based on Hadoop and the last one is for stream processing. (i) Micro benchmark: It evaluates the performance of Hadoop and the file system. (ii) Machine learning: It includes 13 machine learning tasks, e.g., Bayesian Classification, K-means clustering, and Random Forest. (iii) SQL: These tasks perform typical OLAP queries. (iv) Web search. (v) Graph benchmark. (vi) Stream processing: It contains four tasks, including identity, repartition, stateful wordcount, and window aggregation. The identity task reads input data from the message system and then writes it back immediately. The repartition task changes the level of parallelism to measure the performance of data shuffle.

Linear Road Linear Road [8] is a benchmark and accompanying toolkit designed for stream data management systems (SDMS). It aims to measure SDMS performance in processing high-volume streaming and historical data. Linear Road simulates a toll system for motor vehicle expressways in an urban area. The input data is generated by the MIT Traffic Simulator, stored in local files, and directly read by systems. The use cases of Linear Road include toll calculations, accident detection and notification, and historical query processing. There are three types of historical queries: account balance queries, daily expenditure queries, and travel time estimation queries. There are two implementations of Linear Road: one with a commercially available relational database system and the other with a stream data management system (Aurora [1]). In experiments, Linear Road is measured with three metrics: response time, accuracy, and throughput.

3.2 Benchmark Implementation

Benchmarking Scalability of SPSs Deployed Henning et al. [29] evaluate the scalability of various SPSs for cloud-native applications using Theodolite [27]. It also assesses how efficiently systems manage increasing workloads by scaling up resources. They use data generators to produce synthetic data streams and direct the data to Kafka. The evaluation involves five systems: Flink (with both Apache Beam and non-Beam implementations), Kafka Streaming, Hazelcast Jet, and Samza (with Apache Beam implementation), and uses tasks and applications derived from Theodolite.

The experiments are conducted in a private and a public cloud. The public cloud is Google Cloud while the private cloud is from the infrastructure at Kiel University. The authors execute six experiments: (i) Baseline comparison of frameworks: It involves assessing the scalability of systems. (ii) Impact of Apache beam configuration. (iii) Scaling the window aggregation duration. (iv) Scaling on a single node: It evaluates the vertical scaling. (v) Comparing scalability in public and private clouds. (vi) Scaling the cluster size: It evaluates the horizontal scaling. Moreover, this publication provides an overview of the implementations and characteristics of existing stream processing benchmarks.

Evaluation of SPSs for IoT Applications Vikash et al. propose a 4-layer infrastructure to process real-time data from IoT devices [66] using SPSs such as Flink, Storm, Spark Streaming, Apex, and NiFi. The authors evaluate these SPSs using two real-world datasets and employ techniques like RESTful or MQTT to feed data to the systems. They conduct five micro-benchmark experiments with an IoT application: (i) Startup time. (ii) Response time (including processing time and latency). (iii) Throughput. (iv) Jitter. (v) Scalability analysis: It evaluates the throughput with varying cluster sizes.

DSPBench DSPBench [12] is a benchmark specifically designed for distributed stream processing systems. It introduces 13 real-world applications and 2 synthetic applications, which are spanning various domains. The benchmark offers a low-level API for the unified development of these applications. Applications are written once and can be run on any SPSs as long as the specific adapter components have been developed. DSPBench uses real-world datasets for most applications. However, in instances where real-world datasets are unavailable, synthetic datasets are employed.

The benchmark evaluates Storm and Spark Streaming with two experiments: (i) Latency and throughput analysis: It executes three applications and adjusts the parallelism of each operator, such as source, counter, and sink, to assess the latency and throughput of each system under different parallelism settings. (ii) Resource consumption analysis: It evaluates network utilization, CPU, and memory consumption.

OSPBench OSPBench [18, 19] focuses on four popular systems: Flink, Spark Streaming, Structured Streaming, and Kafka Streaming. The benchmark utilizes real-world datasets from the IoT domain, generated by traffic sensors. These datasets are sent to the processing systems via Kafka. They suggest using highly optimized configurations and tuning the parameters separately for each task to ensure accurate performance evaluations. There are four experiments: (i) Latency measurement. (ii) Sustainable throughput measurement [41]. (iii) Burst workload: It evaluates the ability of systems to handle bursts or catch up on delays. (iv) Periodic burst workload. There is only one task pipeline but with varying complexities: ingesting data from Kafka, parsing and joining of data, and performing window aggregation. OSPBench evaluates systems using four metrics: latency, throughput, CPU utilization, and memory utilization.

Based on OSPBench, Van Dongen et al. [21] discuss the fault tolerance and recovery mechanisms of SPSs and conduct three new experiments. (v) Master Failure: It explores two scenarios: one with a single master and another with a high-availability setup using Zookeeper. It measures the impact of master failures on system performance. (vi) Spark Driver Failure: Targeting the failure of the spark driver component within Spark Streaming and Structured Streaming. (vii) Worker Failure: It kills the work node in the middle of the execution and then immediately brings it back up. Moreover, Van Dongen et al. [20] designed a new experiment to analyze the scalability. (viii) Scalability: It includes two scenarios: horizontal scaling with additional workers and vertical scaling with increased CPUs and memory. They introduce three new metrics: garbage collection, network utilization, and filesystem and disk I/O. Since all systems are implemented in Java, they all involve garbage collection. This metric monitors collection time, collection counts, and memory usage during collections. The filesystem and disk I/O metric monitors filesystem usage and calculates bytes read from and written to disk.

An Experimental Evaluation Shahverdi et al. [59] compare the performance of five popular SPSs: Storm, Flink, Structured Streaming, Kafka Streams, and Hazelcast Jet. The systems ingest synthetic data from Kafka and output results to Redis. Two micro-benchmark experiments are executed based on the YSB use case. The first one measures latency with varying data generation rates and number of windows. The second one measures CPU and memory usage.

An Experimental Survey Inoubli et al. [36] discuss the features of both batch processing and stream processing systems. The authors categorize systems based on six key features: programming model, programming languages, type of data sources, iterative data processing, machine learning libraries, and fault tolerance strategy. They evaluate systems in two modes: batch mode and stream mode.

The stream mode evaluates Flink, Storm, Spark Streaming, and Samza using only one application, which analyzes tweets from Twitter. The systems ingest data from Kafka, parse it, and select relevant data for outputting. The dataset is collected from Twitter. Four metrics are used for evaluation: CPU usage, memory usage, disk I/O usage, and bandwidth consumption (i.e., network overhead).

Performance Analysis of Large-Scale SPSs Truong et al. [65] focus on understanding the performance, back-pressure, and expected utilization of Heron[44]. The authors propose an algorithm based on queuing theory to predict the throughput and latency of stream data processing while ensuring system stability. The application generates events with random words from a predefined word set. It measures the throughput and time of backpressure to study the ability to utilize available resources effectively. It also evaluates effective utilization in scaling. Through experiments, the publication identifies the optimal configuration that maximizes effective throughput while minimizing resource consumption.

Scalability and State Yang et al. [71] evaluate SPSs with two applications. The first one assesses Storm with a Java-based trend detection application. This application, based on Twitter's trend detection algorithm, ingests data via Kafka and utilizes real-world Wikipedia data. The Wikipedia data is sourced from the Amazon public web service dataset. The second one evaluates Flink, Storm, and Spark Streaming using the Yahoo Stream Benchmark. Both applications measure the CPU utilization of systems.

RIoTBench RIoTBench [60, 61] is a benchmark for evaluating SPSs in the Internet of Things (IoT) environment. It offers a collection of realistic IoT tasks and applications that can be customized and adjusted to evaluate the performance and scalability of SPSs. RIoTBench contains four real-world IoT streams: Sense Your City, Personal Fitness, Smart Grid, and Taxi Cab Services. These streams are all related to smart city domains, varying in terms of sensor count, attributes per message, message size, and scaling factors used.

RIoTBench includes 27 common IoT tasks, used by four applications: (i) Extract-Transform-Load (ETL): It involves extracting data from various sources, transforming it into a consistent format, and persisting it to table storage, e.g., databases or data warehouses. The purpose of ETL is to ensure that data from different sources can be effectively combined and analyzed. (ii) Statistical summarization (STATS): This category aggregates and analyzes data to provide a high-level overview of the system. Tasks within STATS include finding approximate counts, identifying skewness in data distribution, and using linear regression for online trend analysis. (iii) Model training (TRAIN): TRAIN involves machine learning algorithms, and its results employed in PRED. (iv) Predictive analysis (PRED): It enables early detection of issues and optimization of system performance. RIoTBench evaluates all tasks based on Apache Storm using five metrics: latency, throughput, jitter, CPU utilization, and memory utilization. Jitter refers to the difference between the observed output throughput and the expected output throughput. Additionally, Nasiri et al. [48] extend RIoTBench to include Flink and Spark Streaming. It introduces a new application:

the advertising application sourced from Yahoo Streaming Benchmarks [16]. All SPSs are evaluated based on four metrics: latency, throughput, CPU Usage, and Network Utilization.

A Performance Benchmark for NetFlow Data Čermák et al. [15] focus on effective network flow analysis. They evaluate Storm, Spark Streaming, and Samza with the dataset comprising a network traffic sample from CAIDA [13], sourced from real-world network data. The application has six tasks: identify, filter, count, aggregation, top N, and SYN DoS. The SYN DOS is to detect network attacks represented by a large volume of traffic consisting only of TCP SYN packets from a single source IP address. In experiments, the authors change the number of CPU cores to evaluate the throughput of SPSs. Initially, there is one machine with 32 cores, then they decrease cores or add new machines.

A Performance Comparison Lopez et al. [45] assess SPSs such as Flink, Storm, and Spark Streaming in a threat detection application. The data is from real-world datasets and transmitted to the systems via Kafka. There are two experiments: one measures system throughput with varying parallelism, while the other assesses throughput and message loss rate during node failure.

Stream Processing Systems Benchmark Wang et al. [69] evaluate the performance of Flink, Strom, and Spark Streaming. They introduce a common API to enable different SPSs to use the same set of operations and tasks. They also design generators to produce synthetic data with configurable parameters. These generators send data to Kafka. There are three tasks and two performance metrics: throughput and latency. (i) Workcount: The offline one measures throughput, while the online one measures latency. (ii) AdvClick: It performs the join operator on two streams. (iii) K-means.

StreamBench StreamBench [46] is an end-to-end benchmark for distributed SPSs. It aims to measure performance, fault tolerance ability, and durability under different data scales with Storm and Spark Streaming. StreamBench has two real-world datasets: one for text data and another for numerical data. The data generators and benchmarks run on separate clusters, so the data are sent via the network. As the data publishing speed is faster than the consumption speed and the message system is designed to serve data faster than the consumption speed, StreamBench can evaluate the boundary of SPSs.

StreamBench evaluates seven benchmark tasks with four experiments: (i) Performance workload: It evaluates SPSs with different data scales. (ii) Multi-Recipient performance workload: It introduces the reception ability to represent the proportion of nodes in the computation cluster that receive input data. (iii) Fault tolerance workload: It considers one node failure and measures the fault tolerance ability of systems. (iv) Durability workload: It measures SPSs over a period of time. In addition to throughput and latency, StreamBench also introduces throughput penalty factor and latency penalty factor. These metrics measure the impact of node failure and the percentage of time during which the systems remain available, Qian et al. [53] extend StreamBench to incorporate Apache Storm Trident and Samza.

3.3 Benchmark Evaluations

ShuffleBench ShuffleBench [31] is designed for evaluating large-scale shuffle operations, meeting the requirements for near real-time analytics in a large cloud observability platform. It extends the capabilities of Theodolite [27] to support automated execution in Kubernetes-based cloud environments. It supports implementations for four SPSs: Flink, Kafka Streaming, Hazelcast Jet, and Structured Streaming. ShuffleBench inputs data from load generators. These generators produce data based on configurable parameters and output it to Kafka. It evaluates systems based on throughput (including ad-hoc throughput and sustainable throughput), latency, and scalability. Ad-hoc throughput is to measure throughput when the data volume matches or exceeds the processing rate.

ShuffleBench executes a MapReduce-like architecture task and conducts six micro-benchmark experiments. (i) Throughput. (ii) Latency. (iii) Deployment impact: There are two developments, one has nine instances with one core for each, and the other has three instances with three core for each. It aims to compare systems with different deployments. (iv) Record size impact: It evaluates SPSs with different data record sizes. (v) Consumer count impact. (vi) Selectivity impact.

Streaming vs. functions Pfandzelter et al. [51] present an evaluation study on cloud event processing deployments, focusing on both stateful and stateless applications. The primary aim is to provide decision-making guidelines for application developers from the perspective of cost. The applications and data generators are derived from the Theodolite [27] and use Kafka to pass data. One use case involves stateless storage with Function as a Service (FaaS) platforms like Google Cloud Functions, while the other performs stateful sliding window aggregation with Flink. In experiments, the authors measure FaaS and Flink with cost in different scenarios.

Maximum Sustainable Throughput Evaluation Chu et al. [17] discuss the evaluation of the maximum sustainable throughput (MST) of distributed stream processing systems. The authors propose an adaptive MST evaluation method that incorporates a data-growth factor function. It dynamically adapts the data rate based on latency feedback and achieves accurate and efficient MST evaluation with lower error. They evaluate Storm, Flink, and Heron with four tasks sourced from Yahoo Streaming Bench [16], BigDataBench[68], and HiBench. In experiments, the authors measure systems with execution time, throughput, and error rates (i.e., whether MST is reached).

Analyzing Efficient Stream Processing on Modern Hardware Zeuch et al. [75] analyze and optimize SPSs for effectively utilizing modern hardware. The authors discuss current systems, identify inefficiencies and bottlenecks, and explore data-related and processing-related optimizations for SPSs on modern hardware. They investigate the use of multi-core CPUs, in-memory processing, and emerging network technologies such as RDMA and Infiniband. Additionally, they discuss two execution models (Operator Fusion) for performing data-parallel pipelined stream processing and two parallelization strategies (Operator Fission) for distributing processing among different units. Moreover, they propose a lock-free windowing mechanism aimed at minimizing contention between worker threads.

The evaluation includes seven systems: Flink, Storm, Spark Streaming, Streambox [47], Saber [42], as well as `C++` prototype and Java prototype. Streambox and Saber are two research prototypes, while the `C++` prototype and Java prototype are hand-coded by the authors. The authors select use cases from the Yahoo Streaming Benchmark [16], Linear Road Benchmark [8], and queries from NYC Taxi [58]. The Yahoo Streaming Benchmark and Linear Road employ synthetic data, while NYC Taxi uses real-world data. There are six experiments: (i) End-to-End throughput. (ii) Execution time breakdown: It breaks down the execution time for different CPU components to identify bottlenecks. (iii) Analysis of resource utilization, (iv) Comparison with state-of-the-art approaches, (v) RDMA and Infiniband. (vi) Latency. The evaluated metrics include throughput, latency, and execution time.

Quantitative Impact Evaluation Hesse et al. [32] discusses the performance of distributed SPSs using Apache Beam. They evaluate three systems, including Flink, Spark Streaming, and Apex. It processes four queries, ingesting data from Kafka sourced from a real-world dataset. In experiments, they measure the execution time of systems focusing on three aspects: average execution time, standard deviation of execution time, and the impact of Apache Beam.

Benchmarking Distributed SPSs Karimov et al. [41] perform an evaluation to accurately measure the performance of SPSs, including Flink, Storm, and Spark Streaming. To mitigate potential interference from data exchange between message brokers and systems, the authors design generators to produce data on the fly instead of relying on a message broker or a file system. There is a queue between systems and data generators to ensure isolation and accurate measurement. They propose event-time latency, processing latency, maximum throughput, and sustainable throughput as metrics to accurately measure systems. Event-time latency measures the end-to-end time for events in the system. Processing latency measures the processing time for events. Maximum throughput refers to the highest load of events the system can handle, while sustainable throughput refers to the highest load of events the system can handle without prolonged backpressure. The network and CPU usage are measured as well.

The application is designed in the context of online video games and evaluated with synthetic data. It contains nine experiments: (i) Windowed aggregations. (ii) Windowed joins. (iii) Queries with large windows. (iv) Data skew. (uneven data distribution) (v) Fluctuating workloads. (fluctuating load of events) (vi) Event-time vs. Processing-time latency. (vii) Observing backpressure. (viii) Throughput graphs. (ix) Resource usage statistics. (network and CPU usage)

3.4 Benchmark Tools

SPBench SPBench [23] is a tool designed for creating benchmarks of SPSs. It focuses on various Parallel Programming Interfaces (PPIs) such as Intel TBB [67], FastFlow [2], and SPar [25]. It provides both an Application Programming Interface (API) and a command-line interface (CLI), allowing users to access, modify, and create custom versions of benchmarks. The benchmark tool offers four real-world applications: (i) Bzip2: A data compression application. (ii) Lane detection: Detection of lanes on a road using computer vision techniques. (iii) Person recognition: Recognition and identification of people's faces in images or videos. (iv) Ferret: Similarity search in large databases. Four metrics can be measured, including throughput, latency, CPU usage, and memory usage.

Towards a High-Level Description Pagliari et al. [50] introduce a high-level description model called the Workflow Schema. It is designed for the easy and flexible configuration of benchmark applications. This model comprises a data stream section and a workflow section, enabling the generation of benchmark workloads and data generators. The workflow section allows users to configure the structure and characteristics of the workloads, including parameters such as the number of tasks (depth), task parallelism (scalability), topology shape (connection), task processing load (workload), and message reliability. The stream section contains configuration parameters for data generators, currently supporting only synthetic data generation. These parameters include defining data size, the number of values in each event, value distribution, event distribution, and event generation rate. The authors also present a prototype that generates benchmark applications for Flink, evaluating Flink performance using throughput and latency metrics.

4 Related Work

As stream processing is widely used in various domains, many research efforts focus on SPSs and provide surveys on the state-of-the-art. Isah et al. [38] offer a comparative study of SPSs, including both open-source and commercial systems. Fragkoulis et al. [22] classify stream processing systems into three generations and give an outlook of future stream processing engines. However, both efforts primarily discuss SPSs.

Bajaber et al. [9] discuss various benchmark approaches and tools that have been developed for different categories of big data systems but the SPS benchmarks are not the primary focus. The recent benchmark survey effort by Ihde et al. [35] covers benchmarks for big data systems, high-performance computing, and machine learning, with limited mention of stream processing system benchmarks. There are also many survey efforts focused on specific areas of benchmarks, such as graph processing [11] and machine learning [56, 77]. The literature [29, 31, 41] presented in our paper includes surveys of stream processing benchmarks, but conducting surveys is not their primary focus. In summary, to the best of our knowledge, there is no survey on stream processing system benchmarks with comparable coverage or depth to this work.

5 Conclusion and Discussion

In this paper, we review state-of-the-art stream processing benchmarks, highlighting their focus, scenario, and features. We propose five dimensions to classify and analyze existing benchmarks. We provide a comprehensive overview of every benchmark presented in this paper aim to aid researchers and developers in understanding these benchmark efforts. Additionally, we discuss the challenges and limitations of current stream processing benchmarks and explore potential future directions in the following:

Benchmark Tools Existing efforts [12, 50] can generate benchmarks based on user configurations. However, they are limited to specific stream processing systems, e.g., Flink. There is a need for benchmark tools to use a unified descriptive model like Apache Beam. In this case, they can translate user-defined benchmark workloads into implementations for more stream processing systems. Additionally, benchmark tools with graphical interfaces can help users conduct benchmarks efficiently.

Edge Computing To reduce network costs and improve resource utilization, state-of-the-art approaches [10, 73, 76] shift computations from the central node to machines closer to data sources. However, current benchmarks [61, 66] focus on central processing, where data is ingested to a central node and is processed there. Future benchmarks should incorporate setups that allow conducting experiments on less powerful machines to simulate IoT-like setups.

Research Prototype Most benchmark efforts focus on popular stream processing systems such as Flink, Spark Streaming, and Storm. Some evaluation efforts involve research prototypes, e.g., Zeuch et al. [75] compare performance between these systems and two research prototypes, i.e., Streambox and Saber. Also, many new research prototypes present significant improvements, e.g., LightSaber [62], Scotty [64], Desis [73] or NebulaStream [76]. However, there is still no benchmark available for comparing these prototypes.

References

1. Abadi, D.J., Carney, D., Çetintemel, U., Cherniack, M., Convey, C., Lee, S., Stonebraker, M., Tatbul, N., Zdonik, S.B.: Aurora: a new model and architecture for data

stream management. VLDB Journal **12**(2), 120–139 (2003)

2. Aldinucci, M., Danelutto, M., Meneghin, M., Torquati, M., Kilpatrick, P.: Efficient streaming applications on multi-core with FastFlow: the biosequence alignment test-bed. In: Parallel Computing: From Multicores and GPU's to Petascale, Proceedings of the conference ParCo. Advances in Parallel Computing, vol. 19, pp. 273–280 (2009)
3. Apache Software Foundation: Apache Apex (2018), `https://apex.apache.org`, Online, accessed 2024-06-07
4. Apache Software Foundation: Apache Beam Documentation (2024), `https://beam.apache.org/documentation/`, Online, accessed 2024-06-07
5. Apache Software Foundation: Apache Kafka (2024), `https://kafka.apache.org/`, Online, accessed 2024-06-07
6. Apache Software Foundation: Apache NiFi Documentation (2024), `https://nifi.apache.org/documentation/`, Online, accessed 2024-06-07
7. Apache Software Foundation: Apache Spark: Structured Streaming Programming Guide (2024), `https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html`, Online, accessed 2024-06-07
8. Arasu, A., Cherniack, M., Galvez, E.F., Maier, D., Maskey, A., Ryvkina, E., Stonebraker, M., Tibbetts, R.: Linear Road: A stream data management benchmark. In: Proceedings of the International Conference on Very Large Databases (VLDB). pp. 480–491 (2004)
9. Bajaber, F., Sakr, S., Batarfi, O., Altalhi, A.H., Barnawi, A.: Benchmarking big data systems: A survey. Computer Communications **149**, 241–251 (2020)
10. Benson, L., Grulich, P.M., Zeuch, S., Markl, V., Rabl, T.: Disco: Efficient distributed window aggregation. In: Proceedings of the International Conference on Extending Database Technology (EDBT). pp. 423–426 (2020)
11. Bonifati, A., Fletcher, G.H.L., Hidders, J., Iosup, A.: A survey of benchmarks for graph-processing systems. In: Graph Data Management, Fundamental Issues and Recent Developments, pp. 163–186. Data-Centric Systems and Applications, Springer International Publishing (2018)
12. Bordin, M.V., Griebler, D., Mencagli, G., Geyer, C.F.R., Fernandes, L.G.L.: DSPBench: A suite of benchmark applications for distributed data stream processing systems. IEEE Access **8**, 222900–222917 (2020)
13. CAIDA: Anonymized Internet Traces 2015 (2015), `https://catalog.caida.org/dataset/passive_2015_pcap`, Online, accessed 2024-06-07
14. Carbone, P., Katsifodimos, A., Ewen, S., Markl, V., Haridi, S., Tzoumas, K.: Apache Flink: Stream and batch processing in a single engine. Data Engineering Bulletin **38**(4), 28–38 (2015)
15. Cermák, M., Tovarnák, D., Lastovicka, M., Celeda, P.: A performance benchmark for netflow data analysis on distributed stream processing systems. In: Proceedings of the Network Operations and Management Symposium (NOMS). pp. 919–924 (2016)
16. Chintapalli, S., Dagit, D., Evans, B., Farivar, R., Graves, T., Holderbaugh, M., Liu, Z., Nusbaum, K., Patil, K., Peng, B., Poulosky, P.: Benchmarking streaming computation engines: Storm, Flink and Spark Streaming. In: Proceedings of the International Parallel and Distributed Processing Symposium Workshops (IPDPS). pp. 1789–1792 (2016)
17. Chu, Z., Yu, J., Hamdulla, A.: Maximum sustainable throughput evaluation using an adaptive method for stream processing platforms. IEEE Access **8**, 40977–40988 (2020)

18. van Dongen, G.: Open stream processing benchmark: an extensive analysis of distributed stream processing frameworks. Ph.D. thesis, Ghent University (2021)
19. van Dongen, G., den Poel, D.V.: Evaluation of stream processing frameworks. IEEE Transactions on Parallel and Distributed Systems **31**(8), 1845–1858 (2020)
20. van Dongen, G., den Poel, D.V.: Influencing factors in the scalability of distributed stream processing jobs. IEEE Access **9**, 109413–109431 (2021)
21. van Dongen, G., den Poel, D.V.: A performance analysis of fault recovery in stream processing frameworks. IEEE Access **9**, 93745–93763 (2021)
22. Fragkoulis, M., Carbone, P., Kalavri, V., Katsifodimos, A.: A survey on the evolution of stream processing systems. VLDB Journal **33**(2), 507–541 (2024)
23. Garcia, A.M., Griebler, D., Schepke, C., Fernandes, L.G.: SPBench: a framework for creating benchmarks of stream processing applications. Computing **105**(5), 1077–1099 (2023)
24. Ghazal, A., Rabl, T., Hu, M., Raab, F., Pöss, M., Crolotte, A., Jacobsen, H.: BigBench: towards an industry standard benchmark for big data analytics. In: Proceedings of the International Conference on Management of Data (SIGMOD). pp. 1197–1208 (2013)
25. Griebler, D., Danelutto, M., Torquati, M., Fernandes, L.G.: SPar: A DSL for high-level and productive stream parallelism. Parallel Processing Letters **27**(1), 1740005:1–1740005:20 (2017)
26. Hazelcast, Inc. : Hazelcast Documentation (2024), `https://docs.hazelcast.com/home/`, Online, accessed 2024-06-07
27. Henning, S., Hasselbring, W.: Theodolite: Scalability benchmarking of distributed stream processing engines in microservice architectures. Big Data Research **25**, 100209 (2021)
28. Henning, S., Hasselbring, W.: Demo paper: Benchmarking scalability of cloud-native applications with Theodolite. In: Proceedings of the International Conference on Cloud Engineering (IC2E). pp. 275–276 (2022)
29. Henning, S., Hasselbring, W.: Benchmarking scalability of stream processing frameworks deployed as microservices in the cloud. Journal of Systems and Software **208**, 111879 (2024)
30. Henning, S., Hasselbring, W., Möbius, A.: A scalable architecture for power consumption monitoring in industrial production environments. In: Proceedings of the International Conference on Fog Computing (ICFC). pp. 124–133 (2019)
31. Henning, S., Vogel, A., Leichtfried, M., Ertl, O., Rabiser, R.: Shufflebench: A benchmark for large-scale data shuffling operations with distributed stream processing frameworks. In: Proceedings of the International Conference on Performance Engineering (ICPE). pp. 2–13 (2024)
32. Hesse, G., Matthies, C., Glass, K., Huegle, J., Uflacker, M.: Quantitative impact evaluation of an abstraction layer for data stream processing systems. In: Proceedings of the International Conference on Distributed Computing Systems (ICDCS). pp. 1381–1392 (2019)
33. Hesse, G., Matthies, C., Perscheid, M., Uflacker, M., Plattner, H.: ESPBench: The enterprise stream processing benchmark. In: Proceedings of the International Conference on Performance Engineering (ICPE). pp. 201–212 (2021)
34. Huang, S., Huang, J., Dai, J., Xie, T., Huang, B.: The HiBench benchmark suite: Characterization of the MapReduce-based data analysis. In: Proceedings of the International Conference on Data Engineering (ICDE) Workshops. pp. 41–51 (2010)
35. Ihde, N., Marten, P., Eleliemy, A., Poerwawinata, G., Silva, P., Tolovski, I., Ciorba, F.M., Rabl, T.: A survey of big data, high performance computing, and machine

learning benchmarks. In: TPC Technology Conference on Performance Evaluation and Benchmarking (TPCTC). Lecture Notes in Computer Science, vol. 13169, pp. 98–118 (2021)

36. Inoubli, W., Aridhi, S., Mezni, H., Maddouri, M., Nguifo, E.M.: An experimental survey on big data frameworks. Future Generation Computer Systems **86**, 546–564 (2018)
37. Intel Corporation: HiBench Suite (2024), `https://github.com/Intel-bigdata/HiBench`, Online, accessed 2024-06-07
38. Isah, H., Abughofa, T., Mahfuz, S., Ajerla, D., Zulkernine, F.H., Khan, S.: A survey of distributed data stream processing frameworks. IEEE Access **7**, 154300–154316 (2019)
39. Ivanov, T., Bedué, P., Ghazal, A., Zicari, R.V.: Adding velocity to BigBench. In: Proceedings of the International Workshop on Testing Database Systems (DBTest). pp. 6:1–6:6 (2018)
40. Janssens, J.: Outlier selection and one-class classification. Ph.D. thesis, Tilburg University (2013), series: Tilburg center for Cognition and Communication (TiCC) Ph.D. Series Volume: 27
41. Karimov, J., Rabl, T., Katsifodimos, A., Samarev, R., Heiskanen, H., Markl, V.: Benchmarking distributed stream data processing systems. In: Proceedings of the International Conference on Data Engineering (ICDE). pp. 1507–1518 (2018)
42. Koliousis, A., Weidlich, M., Fernandez, R.C., Wolf, A.L., Costa, P., Pietzuch, P.R.: SABER: window-based hybrid stream processing for heterogeneous architectures. In: Proceedings of the International Conference on Management of Data (SIGMOD). pp. 555–569 (2016)
43. Kreps, J., Narkhede, N., Rao, J., et al.: Kafka: A distributed messaging system for log processing. In: Proceedings of the NetDB. vol. 11, pp. 1–7 (2011)
44. Kulkarni, S., Bhagat, N., Fu, M., Kedigehalli, V., Kellogg, C., Mittal, S., Patel, J.M., Ramasamy, K., Taneja, S.: Twitter Heron: Stream processing at scale. In: Proceedings of the International Conference on Management of Data (SIGMOD). pp. 239–250 (2015)
45. Lopez, M.A., Lobato, A.G.P., Duarte, O.C.M.B.: A performance comparison of open-source stream processing platforms. In: Proceedings of the Global Communications Conference (GLOBECOM). pp. 1–6 (2016)
46. Lu, R., Wu, G., Xie, B., Hu, J.: Stream Bench: Towards benchmarking modern distributed stream computing frameworks. In: Proceedings of the International Conference on Utility and Cloud Computing (UCC). pp. 69–78 (2014)
47. Miao, H., Park, H., Jeon, M., Pekhimenko, G., McKinley, K.S., Lin, F.X.: Streambox: Modern stream processing on a multicore machine. In: Proceedings of the USENIX Annual Technical Conference (USENIX ATC). pp. 617–629 (2017)
48. Nasiri, H., Nasehi, S., Goudarzi, M.: Evaluation of distributed stream processing frameworks for IoT applications in smart cities. Journal of Big Data **6**, 52 (2019)
49. Noghabi, S.A., Paramasivam, K., Pan, Y., Ramesh, N., Bringhurst, J., Gupta, I., Campbell, R.H.: Samza: Stateful scalable stream processing at LinkedIn. Proceedings of the VLDB Endowment **10**(12), 1634–1645 (2017)
50. Pagliari, A., Huet, F., Urvoy-Keller, G.: Towards a high-level description for generating stream processing benchmark applications. In: Proceedings of the International Conference on Big Data (BigData). pp. 3711–3716 (2019)
51. Pfandzelter, T., Henning, S., Schirmer, T., Hasselbring, W., Bermbach, D.: Streaming vs. Functions: A cost perspective on cloud event processing. In: Proceedings of the International Conference on Cloud Engineering (IC2E). pp. 67–78 (2022)

52. Pöss, M., Nambiar, R.O., Walrath, D.: Why you should run TPC-DS: A workload analysis. In: Proceedings of the International Conference on Very Large Databases (VLDB). pp. 1138–1149 (2007)
53. Qian, S., Wu, G., Huang, J., Das, T.: Benchmarking modern distributed streaming platforms. In: Proceedings of the International Conference on Industrial Technology (ICIT). pp. 592–598 (2016)
54. Rabl, T., Frank, M., Danisch, M., Jacobsen, H., Gowda, B.: The vision of BigBench 2.0. In: Proceedings of the Workshop on Data analytics in the Cloud (DanaC). pp. 3:1–3:4 (2015)
55. Rabl, T., Poess, M., Danisch, M., Jacobsen, H.: Rapid development of data generators using meta generators in PDGF. In: Proceedings of the International Workshop on Testing Database Systems (DBTest). pp. 5:1–5:6. ACM (2013)
56. Reuther, A., Michaleas, P., Jones, M., Gadepally, V., Samsi, S., Kepner, J.: Survey and benchmarking of machine learning accelerators. In: Proceedings of the High Performance Extreme Computing Conference (HPEC). pp. 1–9 (2019)
57. Sax, M.J., Wang, G., Weidlich, M., Freytag, J.: Streams and tables: Two sides of the same coin. In: Proceedings of the International Workshop on Real-Time Business Intelligence and Analytics (BIRTE). pp. 1:1–1:10 (2018)
58. Schneider, T.W.: Analyzing 1.1 Billion NYC Taxi and Uber Trips, with a Vengeance (2018), `https://toddwschneider.com/posts/analyzing-1-1-billion-nyc-taxi-and-uber-trips-with-a-vengeance/`, Online, accessed 2024-06-07
59. Shahverdi, E., Awad, A., Sakr, S.: Big stream processing systems: An experimental evaluation. In: Proceedings of the International Conference on Data Engineering (ICDE) Workshops. pp. 53–60 (2019)
60. Shukla, A., Chaturvedi, S., Simmhan, Y.: RIoTBench: An IoT benchmark for distributed stream processing systems. Concurrency and Computation: Practice and Experience **29**(21) (2017)
61. Shukla, A., Simmhan, Y.: Benchmarking distributed stream processing platforms for IoT applications. In: TPC Technology Conference on Performance Evaluation and Benchmarking (TPCTC). Lecture Notes in Computer Science, vol. 10080, pp. 90–106 (2016)
62. Theodorakis, G., Koliousis, A., Pietzuch, P.R., Pirk, H.: LightSaber: Efficient window aggregation on multi-core processors. In: Proceedings of the International Conference on Management of Data (SIGMOD). pp. 2505–2521 (2020)
63. Toshniwal, A., Taneja, S., Shukla, A., Ramasamy, K., Patel, J.M., Kulkarni, S., Jackson, J., Gade, K., Fu, M., Donham, J., Bhagat, N., Mittal, S., Ryaboy, D.V.: Storm@Twitter. In: Proceedings of the International Conference on Management of Data (SIGMOD). pp. 147–156 (2014)
64. Traub, J., Grulich, P.M., Cuellar, A.R., Breß, S., Katsifodimos, A., Rabl, T., Markl, V.: Scotty: General and efficient open-source window aggregation for stream processing systems. Transactions on Database Systems (TODS) **46**(1), 1:1–1:46 (2021)
65. Truong, M.T., Harwood, A., Sinnott, R.O., Chen, S.: Performance analysis of large-scale distributed stream processing systems on the cloud. In: Proceedings of the International Conference on Cloud Computing, (CLOUD). pp. 754–761 (2018)
66. Vikash, Mishra, L., Varma, S.: Performance evaluation of real-time stream processing systems for internet of things applications. Future Generation Computer Systems **113**, 207–217 (2020)
67. Voss, M., Asenjo, R., Reinders, J.: Pro TBB: C++ parallel programming with threading building blocks. Apress Berkeley, CA (2019)

68. Wang, L., Zhan, J., Luo, C., Zhu, Y., Yang, Q., He, Y., Gao, W., Jia, Z., Shi, Y., Zhang, S., Zheng, C., Lu, G., Zhan, K., Li, X., Qiu, B.: BigDataBench: A big data benchmark suite from internet services. In: Proceedings of the International Symposium on High Performance Computer Architecture (HPCA). pp. 488–499 (2014)
69. Wang, Y.: Stream Processing Systems Benchmark: StreamBench. Master's thesis, Aalto University (2016)
70. Yahoo! Inc.: Yahoo Streaming Benchmarks (2024), `https://github.com/yahoo/streaming-benchmarks`, Online, accessed 2024-06-07
71. Yang, S., Jeong, Y., Hong, C., Jun, H., Burgstaller, B.: Scalability and state: A critical assessment of throughput obtainable on big data streaming frameworks for applications with and without state information. In: Proceedings of the Parallel Processing Euro-Par Workshops. Lecture Notes in Computer Science, vol. 10659, pp. 141–152. Springer (2017)
72. Yi, L., Dai, J.: Experience from Hadoop benchmarking with HiBench: From micro-benchmarks toward end-to-end pipelines. In: Proceedings of the Workshop Series on Big Data Benchmarking (WBDB). Lecture Notes in Computer Science, vol. 8585, pp. 43–48 (2013)
73. Yue, W., Benson, L., Rabl, T.: Desis: Efficient window aggregation in decentralized networks. In: Proceedings of the International Conference on Extending Database Technology (EDBT). pp. 618–631. OpenProceedings.org (2023)
74. Zaharia, M., Das, T., Li, H., Hunter, T., Shenker, S., Stoica, I.: Discretized streams: fault-tolerant streaming computation at scale. In: Proceedings of the Symposium on Operating Systems Principles (SOSP). pp. 423–438 (2013)
75. Zeuch, S., Breß, S., Rabl, T., Monte, B.D., Karimov, J., Lutz, C., Renz, M., Traub, J., Markl, V.: Analyzing efficient stream processing on modern hardware. Proceedings of the VLDB Endowment **12**(5), 516–530 (2019)
76. Zeuch, S., Chaudhary, A., Monte, B.D., Gavriilidis, H., Giouroukis, D., Grulich, P.M., Breß, S., Traub, J., Markl, V.: The NebulaStream platform for data and application management in the internet of things. In: Proceedings of the Conference on Innovative Data Systems Research (CIDR) (2020)
77. Zöller, M., Huber, M.F.: Benchmark and survey of automated machine learning frameworks. Journal of Artificial Intelligence Research **70**, 409–472 (2021)