Doctoral Symposium: A New Application Benchmark for Data Stream Processing Architectures in an Enterprise Context

Guenter Hesse*
Hasso Plattner Institute
August-Bebel-Str. 88
14482, Potsdam, Germany
guenter.hesse@hpi.de

Christoph Matthies
Hasso Plattner Institute
August-Bebel-Str. 88
14482, Potsdam, Germany
christoph.matthies@hpi.de

Benjamin Reissaus
Hasso Plattner Institute
August-Bebel-Str. 88
14482, Potsdam, Germany
benjamin.reissaus@student.hpi.de

Matthias Uflacker
Hasso Plattner Institute
August-Bebel-Str. 88
14482, Potsdam, Germany
matthias.uflacker@hpi.de

ABSTRACT

Against the backdrop of ever-growing data volumes and trends like the Internet of Things (IoT) or Industry 4.0, Data Stream Processing Systems (DSPSs) or data stream processing architectures in general receive a greater interest. Continuously analyzing streams of data allows immediate responses to environmental changes. A challenging task in that context is assessing and comparing data stream processing architectures in order to identify the most suitable one for certain settings.

The present paper provides an overview about performance benchmarks that can be used for analyzing data stream processing applications. By describing shortcomings of these benchmarks, the need for a new application benchmark in this area, especially for a benchmark covering enterprise architectures, is highlighted. A key role in such an enterprise context is the combination of streaming data and business data, which is barely covered in current data stream processing benchmarks. Furthermore, first ideas towards the development of a solution, i.e., a new application benchmark that is able to fill the existing gap, are depicted.

CCS CONCEPTS

•**Information systems** → *Stream management;* Enterprise applications; •**Software and its engineering** → *Software performance;*

KEYWORDS

Performance Benchmarking; Benchmark Development; Data Stream Processing; Stream Processing; Internet of Things

ACM Reference format:
Guenter Hesse, Christoph Matthies, Benjamin Reissaus, and Matthias Uflacker. 2017. Doctoral Symposium: A New Application Benchmark for Data Stream Processing Architectures in an Enterprise Context. In *Proceedings of DEBS '17, Barcelona, Spain, June 19-23, 2017,* 4 pages.
DOI: http://dx.doi.org/10.1145/3093742.3093902

*The PhD student working on the presented project.

DEBS '17, Barcelona, Spain

1 INTRODUCTION

The ever increasing amount of data that is being produced nowadays, from smart factories to the Internet of Things (IoT) or Industry 4.0, gives rise to completely new challenges and opportunities.

A particularly interesting business domain in this context is industrial manufacturing. In a GE battery production plant in New York (state), for example, 10,000 different data attributes are captured, some as often as every 250ms [14]. Furthermore, modern manufacturing equipment, e.g., injection molding machines, generate multiple gigabytes or even up to terabytes of sensor data, daily [7]. This data provides a detailed overview of the current state of machines. It allows timely reactions to events, such as failures. By enriching gathered data with context information, e.g., information about suppliers of certain goods or machine operators, a better understanding of the holistic value chain can be reached. Up-to-date information on product status allows reacting as soon as possible to changing environments, both, locally at machine level, as well as globally, e.g., concerning production order management.

An especially interesting, though not completely new, type of applications that focusses on analyzing high frequency data sources, e.g., sensor data, are Data Stream Processing Systems (DSPSs). In recent years, due to the high demand for Big Data analysis, a multitude of new DSPSs were developed. Examples for these are Apache Flink, Apache Storm, Apache Spark Streaming, Apache Samza, Twitter Heron and Apache Apex [5, 6, 9]. Contrary to such recently developed systems, Aurora [1] and STREAM [3], for instance, were already presented in the early 2000's.

Although a broad variety of systems allows for more choice, picking the system or architecture that best suits a given use case becomes more of an issue. Due to the lack of satisfying real-world application benchmarks for analyzing data streams, alone as well as together with corresponding business data, this is currently a particularly challenging task. We aim to tackle this issue by providing an application benchmark focussed on data stream processing in an enterprise context.

The remainder of this paper is structured as follows: Section 2 presents related work in the area of performance benchmarking for data stream processing and Section 3 highlights the need for a new data stream processing benchmark. Section 4 introduces the idea

for an architecture for the new enterprise streaming benchmark. Section 5 concludes, giving a brief summary and illustrating areas for future work.

2 RELATED WORK

Although Data Stream Processing Systems or data stream processing in general are not a recent development, compared to the area of Database Management Systems, only few benchmarks are available. An overall comparison of the benchmarks presented in the following is shown in Table 1. The mentioned term *system under test* (SUT) stands for "the system to be evaluated" [11], so the system that is responsible for processing incoming data and for responding according to the defined queries.

One of the most, if not the most popular application benchmark focussing on data stream processing is the *Linear Road* Benchmark by Arasu et al. [4]. The benchmark contains a toolkit comprising a data generator, a data sender as well as a result validator. With an execution of a Linear Road implementation, a variable tolling system for a metropolitan area covering multiple expressways is simulated. The amount of tolls is thereby dependent on multiple aspects of the traffic situation on highways.

The data sender emits data records of four different types. Car position reports represent the data type with, by far, most of the input data. The minor part of remaining data records is spread across the other three input types and, contrary to car position reports, express explicit user requests that always expect an answer from the system. Depending on the overall situation on highways, car position reports might require the system to create an output or not.

Overall, the benchmark authors defined four different queries or output types. Due to complexity, the implementation of the lastly presented query was even skipped in the two implementations described in [4]. Besides live data, historical data covering ten weeks of tolling history is generated and partly has to be used in order to produce the correct answer.

With regard to benchmark result metrics, Linear Road defines one overall metric called the L-Rating. The L-Rating indicates how many expressways, a configurable parameter for the data generation step, a system can handle without violating the defined maximum response times for each query.

StreamBench [10] aims at benchmarking distributed DSPSs. It can be categorized as a micro benchmark, i.e., it measures atomic operations, such as the execution of a projection rather than those of more complex applications as in Linear Road. Thus, when a system's performance for real-world scenarios or applications is to be evaluated, micro benchmark results only have limited validity. However, if, e.g., two distinct filter operator implementations are to be compared, micro benchmarks have advantages over application benchmarks due to their simplicity and the fact that measurements only contain the relevant parts without much overhead.

StreamBench defines seven benchmark queries or programs in total. Six of these work with textual data, one uses numerical data as input. Moreover, four out of seven queries contain a single computation step, such as extracting a certain field out of a data record, and three queries comprise multiple computation steps. Four out of seven are stateless and three queries have to maintain a state. Although seven different queries are presented, some typical streaming operations like window functions are not taken into account.

Next to query definition, StreamBench defines four workload suites. Those suites impact the way the benchmark is executed. In particular, the workload suites vary by, e.g., input data scales, executed queries, the existence of an intentional node failure or employed benchmark result metrics.

As input data, StreamBench utilizes two real-world data sets, one with textual and one with numerical data. Although real-world data sets are always desirable since they help increase the benchmark's relevance, the two data sets used in StreamBench only serve as seeds for data generation and thus do not entirely represent reality.

In contrast to Linear Road, StreamBench employs a message broker, which is used for decoupling data generation and consumption. This approach is similar to the one proposed in this paper, as described in Section 4. For the results presented in [10], Apache Kafka [8] is used as broker, which is again similar the benchmark described in the present paper. A tool for data ingestion released in the context of StreamBench was not described by the authors.

Dependent on the workload suite, StreamBench defines the metrics throughput, the average count per second and the data size in bytes per second, both in total and per node, as well as latency. Additionally, the authors introduce three new metrics: a durability index (uptime), a throughput penalty factor (assessing throughput change for node failure), and a latency penalty factor (assessing latency change for node failure). To the best of our knowledge, result validation with respect to query outcome is not supported by a dedicated tool.

RIoTBench, a benchmark by Shukla et al. [12], focuses on benchmarking distributed DSPS. It defines several micro benchmark scenarios as well as four application benchmark use cases, which represent combined micro benchmarks. In particular, these cover Extract, Transform and Load (ETL) processes, statistics generation, model training as well as a predictive analytics scenarios.

With respect to data, RIoTBench uses scaled real-world data sets from different IoT domains, namely, smart city, smart energy and health. A dedicated data sender for ingesting data into the system or a query result validation tool are not provided.

Next to latency, throughput as well as CPU and memory utilization, RIoTBench measures *jitter* as a metric, which is defined as the difference between expected and actual output rate during a certain interval.

3 SHORTCOMINGS IN EXISTING BENCHMARKS

In summary, we see the need for a new streaming benchmark for several reasons. First, currently only two major application benchmarks for data stream processing exist and only one of them considers the distribution of systems in its metrics.

Second, historical and transactional data is not or only barely taken into account in all of the presented benchmarks, which we believe is a crucial aspect in many enterprise contexts in order to achieve the greatest added value. Only when combining streaming data, e.g., sensor measurements from manufacturing equipment, with its corresponding business information, one is able to not only

Table 1: Overview of Data Stream Processing System Benchmarks

Criteria \ Benchmark	Linear Road	StreamBench	RIoTBench
Benchmark Type	Application benchmark	Micro benchmark	Micro benchmark and application benchmark
Considered System Under Test	DSPS or DBMS	Distributed DSPS	Distributed DSPS
Domain	Smart City (variable tolling)	Log processing and network traffic monitoring	Smart City, Smart Energy, Health (in general - IoT)
Data	Synthetic data (including historical data)	Synthetic data (real-world data used as seed)	Scaled real-world data sets
Metrics	One self-defined metric (throughput under latency restriction; hardware-independent): L-Rating	Throughput or throughput-related (different forms), latency or latency-related (different forms), system availability	Latency, throughput, jitter (difference between expected and actual output rate), resource utilization
Query Result Validation	Validation tool provided	Not considered	Not considered

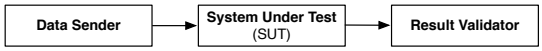

Figure 1: Simplified Benchmark Architecture in Fundamental Modeling Concepts (FMC)

react locally, but also globally, as mentioned in Section 1. As such, questions relating to business use cases, including interfaces or efficient combination of live and historical data, are currently challenging to answer. Additionally, the majority of current streaming benchmarks lack tool support, e.g., for result validation or data ingestion, which complicates implementing these benchmarks and retrieving objective results.

4 AN ARCHITECTURE FOR AN ENTERPRISE STREAMING BENCHMARK

Figure 1 depicts a simplified overview of a general architecture for benchmarks with focus on data stream processing, which helps getting an understanding of the general benchmark process. It shows three main components: the *data sender*, *system under test* (SUT) and the *result validator*.

The data sender is responsible for ingesting data into the SUT. The SUT is processes incoming data and responds according to the defined queries. Produced results are evaluated by the result validator. This component could also calculate benchmark metrics, e.g., related to performance, which might already be monitored when running the benchmark.

A more detailed overview of the proposed architecture for an enterprise streaming benchmark is illustrated in Figure 2.

The *input data* as the first component, i.e., one or more files in CSV or similar format containing sensor data from a manufacturing context, are the input for the system with respect to streaming data. This data will be, in the best case, entirely real-world data. If no suitable data set can be identified, a data generator will be needed that takes care of creating the input data. The preferred way of doing so is the generation based on an existing real-world data set in order to keep characteristics.

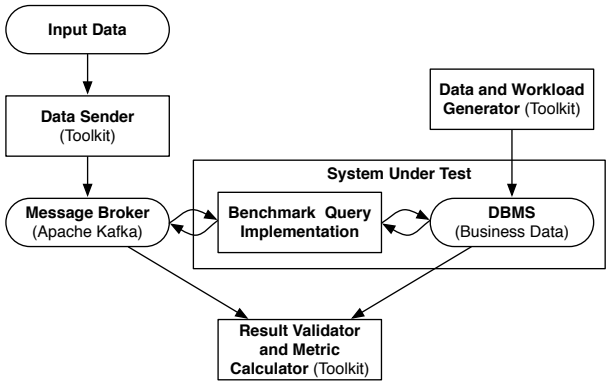

Figure 2: Architecture of the Enterprise Streaming Benchmark in FMC

The input data is read and forwarded by the *data sender*. Sending rate, i.e., the number of messages that are forwarded per second, shall be configurable. That allows analyzing different scenarios and environments with respect to the input rate the SUT has to handle. Although it is closer to reality to send all records according to the timestamp they may contain, this behavior might not be sufficient to satisfy configured data ingestion rates. When solely relying on existing timestamps, it might not be possible to test, e.g., how much throughput a system can reach or how a system would behave with a doubled number of input records per second.

The *message broker* acts as interface between data sender and SUT and as storage for query results in the defined setup. In the presented benchmark architecture, Apache Kafka [8] in particular shall act as message broker. One reason for using Apache Kafka is its usage in enterprise software architectures. An existing field of application for Kafka is, e.g., as interface to a DSPS. Thus, it reflects

reality and so adds relevance to the benchmark. Such usages of Kafka in combination with a DSPS were presented by, e.g., Bouygues Telecom [2] and Zalando [13].

Additional reasons for choosing Kafka are related to scalability and benchmark metrics calculations, i.e., we aim to leverage Kafka timestamp functionality for calculating processing times. Using Kafka timestamps allows to stay independent from implementations of SUTs and thus, system-dependent differences in terms of time measurements can be preempted. Moreover, Kafka already provides further configuration parameters, which can be used by the data sender, e.g., for adjusting the mentioned throughput. So it is possible to send data records in batches with configurable size or to adjust the level of acknowledgements used when sending data.

The *SUT* comprises two main components: the query implementation and a DBMS. The implementation represents the queries defined by the benchmark and can be done using any technology, e.g., a DSPS or DBMS features such as stored procedures. The only requirement is the ability to communicate with a DBMS and with the message broker, i.e., with Apache Kafka. Historical or business data is consumed from a DBMS on demand, i.e., whenever a query requires this data. As the speed of DBMS can influence the responsive time of queries, it is part of the SUT. Some queries might require updating one or multiple historical data records, which is why a bi-directional connection between the query implementation and the historical data store exists. Except for such business data updates, query results are returned to the message broker.

The *data and workload generator* simulates realistic usage through inserting business data and executing analytical queries on the DBMS containing the historical data. By doing so, a real-world environments can be simulated.

The last component illustrated in Figure 2 is the *result validator and benchmark metrics calculator*. It reads the query output from the message broker as well as from the DBMS containing the business data and checks the correctness of results. Additionally, the benchmark results, i.e., the scores for the benchmark metrics, are calculated.

5 CONCLUSIONS

Within the present paper, related work and the need for a new application benchmark for data stream processing in an enterprise context is presented. Furthermore, the concept for such a new benchmark is illustrated. The proposed benchmark focusses on industrial manufacturing as domain and provides a toolkit for data ingestion into the SUT as well as query result validation and benchmark metrics calculation. While some queries can be answered solely using streaming data, other queries require access to historical or business data in order to produce correct results.

By developing the benchmark, we aim to fill the gap that exists in the area of benchmarking enterprise architectures with focus on data stream processing.

The next steps are to validate the presented architecture by discussing it with industry partners as well as to create a first minimal viable benchmark implementation. That includes the development of an initial query list. Moreover, a set of benchmark metrics shall be defined that are able to assess relevant aspects, especially for the enterprise context.

REFERENCES

[1] Daniel J. Abadi, Don Carney, Ugur Çetintemel, Mitch Cherniack, Christian Convey, Sangdon Lee, Michael Stonebraker, Nesime Tatbul, and Stan Zdonik. 2003. Aurora: A New Model and Architecture for Data Stream Management. *The VLDB Journal* 12, 2 (Aug. 2003), 120–139. https://doi.org/10.1007/s00778-003-0095-z

[2] Mohamed Amine Abdessemed. 2015. Real-time Data Integration with Apache Flink & Kafka @Bouygues Telecom. http://www.slideshare.net/FlinkForward/mohamed-amine-abdessemed-realtime-data-integration-with-apache-flink-kafka. (2015). Accessed: 2017-04-06.

[3] Arvind Arasu, Brian Babcock, Shivnath Babu, Mayur Datar, Keith Ito, Itaru Nishizawa, Justin Rosenstein, and Jennifer Widom. 2003. STREAM: The Stanford Stream Data Manager (Demonstration Description). In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data (SIGMOD '03)*. ACM, New York, NY, USA, 665–665. https://doi.org/10.1145/872757.872854

[4] Arvind Arasu, Mitch Cherniack, Eduardo Galvez, David Maier, Anurag S. Maskey, Esther Ryvkina, Michael Stonebraker, and Richard Tibbetts. 2004. Linear Road: A Stream Data Management Benchmark. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30 (VLDB '04)*. VLDB Endowment, 480–491. http://dl.acm.org/citation.cfm?id=1316689.1316732

[5] T. Dunning and E. Friedman. 2016. *Streaming Architecture: New Designs Using Apache Kafka and MapR Streams*. O'Reilly Media.

[6] Guenter Hesse and Martin Lorenz. 2015. Conceptual Survey on Data Stream Processing Systems. In *Proceedings of the 2015 IEEE 21st International Conference on Parallel and Distributed Systems (ICPADS) (ICPADS '15)*. IEEE Computer Society, Washington, DC, USA, 797–802. https://doi.org/10.1109/ICPADS.2015.106

[7] Marco F. Huber, Martin Voigt, and Axel-Cyrille Ngonga Ngomo. 2016. Big data architecture for the semantic analysis of complex events in manufacturing. In *Informatik 2016, 46. Jahrestagung der Gesellschaft für Informatik, 26.-30. September 2016, Klagenfurt, Österreich*. 353–360. http://subs.emis.de/LNI/Proceedings/Proceedings259/article173.html

[8] Jay Kreps, Neha Narkhede, Jun Rao, et al. 2011. Kafka: A distributed messaging system for log processing. In *SIGMOD Workshop on Networking Meets Databases*.

[9] 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)*. ACM, New York, NY, USA, 239–250. https://doi.org/10.1145/2723372.2742788

[10] Ruirui Lu, Gang Wu, Bin Xie, and Jingtong Hu. 2014. Stream Bench: Towards Benchmarking Modern Distributed Stream Computing Frameworks. In *Proceedings of the 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing (UCC '14)*. IEEE Computer Society, Washington, DC, USA, 69–78. https://doi.org/10.1109/UCC.2014.15

[11] D. A. Menasce. 2002. TPC-W: a benchmark for e-commerce. *IEEE Internet Computing* 6, 3 (May 2002), 83–87. https://doi.org/10.1109/MIC.2002.1003136

[12] Anshu Shukla, Shilpa Chaturvedi, and Yogesh Simmhan. 2017. RIoTBench: A Real-time IoT Benchmark for Distributed Stream Processing Platforms. *CoRR* abs/1701.08530 (2017). http://arxiv.org/abs/1701.08530

[13] Mihail Vieru and Javier López. 2016. Flink in Zalando's World of Microservices. http://www.slideshare.net/ZalandoTech/flink-in-zalandos-world-of-microservices-62376341. (2016). Accessed: 2017-04-06.

[14] Steven Weiner and David Line. 2014. Manufacturing and the data conundrum - Too much? Too little? Or just right? https://www.eiuperspectives.economist.com/sites/default/files/Manufacturing_Data_Conundrum_Jul14.pdf. (2014). Accessed: 2017-03-01.