Senska –
Towards an Enterprise Streaming Benchmark

Guenter Hesse, Benjamin Reissaus, Christoph Matthies, Martin Lorenz, Milena Kraus, and Matthias Uflacker

Hasso Plattner Institute
University of Potsdam
14482 Potsdam, Germany
{firstname.lastname}@hpi.de

Abstract. In the light of growing data volumes and continuing digitization in fields such as Industry 4.0 or Internet of Things, data stream processing have gained popularity and importance. Especially enterprises can benefit from this development by augmenting their vital, core business data with up-to-date streaming information. Enriching this transactional data with detailed information from high-frequency data streams allows answering new analytical questions as well as improving current analyses, e.g., regarding predictive maintenance. Comparing such data stream processing architectures for use in an enterprise context, i.e., when combining streaming and business data, is currently a challenging task as there is no suitable benchmark.

In this paper, we give an overview about performance benchmarks in the area of data stream processing. We highlight shortcomings of existing benchmarks and present the need for a new benchmark with a focus on an enterprise context. Furthermore, the ideas behind *Senska*, a new enterprise streaming benchmark that shall fill this gap, and its architecture are introduced.

Keywords: Benchmarking, Benchmark Development, Data Stream Processing, Stream Processing, Internet of Things

1 Introduction

Due to the ever increasing velocity and volume of data that is being produced nowadays, completely new challenges and opportunities arise.

Terms like Smart Factories, Industry 4.0, and Internet of Things (IoT) have gained traction to describe some of such new developments which bring new possibilities in how business can be done.

Industrial manufacturing is a particularly interesting domain in this context. An example for a factory where a high volume of data is captured with high velocity is the GE battery production plant in New York (state). There are 10,000 different data attributes recorded, some as often as every 250ms [23]. Modern

The final publication is available at Springer via http://dx.doi.org/10.1007/978-3-319-72401-0_3.

manufacturing equipment, e.g., injection molding machines, can generate up to terabytes of sensor data, daily [13]. Such data provides detailed information about the current state of machines and allows timely reactions to events, such as failures or changes in environment. When it comes to unlocking further efficiency improvements through IoT technologies such as sensors, highly-optimized production facilities are one of the key areas [17]. Combining gathered IoT data with existing transactional or business data, e.g., supplier information or information about machine operators, can lead to a better understanding of the holistic value chain. This combination of machine or sensor data and business data, which allows answering new analytical questions or existing ones in greater detail, can be described as vertical integration. A practical example of applying these ideas would be a printing machine, where the humidity is regulated depending on current sensor measurements (streaming data) as well as the currently used colors and paper (business or historical data) in order to optimize print quality.

Compared to transactional data, IoT or sensor data, as examples of streaming data, differ in aspects such as velocity and volume. A brief comparison of both these kinds of data is shown in Table 1. In order to efficiently handle sensor and general data streams as well as their analysis, new technologies were created.

Table 1. Comparison of sensor data and business data

Characteristic	Sensor Data	Business Data
Volume and Velocity	Up to multiple terabytes by a single machine, daily [13]	Multiple terabytes in total, e.g., for a 20 years old SAP ERP installation at a leading Canadian energy company [21]
Data Quality	Measurement errors, lost data	Correctness crucial for business
Data Manipulations	No updates	Updates exist
References	Strong time and location reference	Strong business process reference
Value for Enterprises	Usually not crucial for daily business	Essential for daily business

A particularly interesting example for a group of systems that can be leveraged for analyzing high frequency data sources are Data Stream Processing Systems (DSPSs). These systems analyze streams of data on the fly using continuous queries. Therefore, the generation of output is dependent on the underlying data streams, i.e., on the arrival of new data points. Moreover, the order of incoming data records is considered, meaning a potential out-of-order arrival at the DSPS may need to be handled in order to produce correct results. Compared to traditional Database Management Systems (DBMSs), the concepts employed in DSPSs differ in some aspects, e.g., with respect to queries, which usually do not run continuously on DBMSs. Storing data only for as long as it is needed for analysis not only benefits performance and data throughput, but also saves storage costs. As an extension to DBMSs, those benefits of DSPSs can be leveraged in enterprise applications. When analyzing data streams in DSPSs, data can be combined with data from business application databases, allowing for new business ideas and far-reaching optimizations of existing processes.

A multitude of new DSPSs were developed in recent years, such as Apache Flink, Apache Storm, Apache Spark Streaming, Apache Samza, Twitter Heron and Apache Apex [11, 15, 8]. Contrary to these recently developed systems, Aurora [3] and STREAM [5], 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. As shown, there is already a wide choice in the group of DSPSs, whose usage represents just one way of analyzing data streams. An alternative approach could be storing data streams in a database and analyzing them afterwards.

Due to the lack of satisfying real-world application benchmarks assessing data stream processing architectures, including the combination of streaming and transactional data for analyses, this is currently a certainly challenging task. We aim to tackle this issue by developing an application benchmark focussed on data stream processing architectures in an enterprise context, *Senska* - an Enterprise Streaming Benchmark (ESB). In this paper we present the following contributions:

- Illustration and motivation of the need for a new application benchmark for data stream processing
- The design objectives of Senska and their underlying concepts
- A first draft of the overall Senska architecture - design decisions are explained and selected components are presented in more detail

The remainder of this paper is structured as follows: Section 2 presents related work in the area of benchmarking and highlights the need for a new data stream processing benchmark. Section 3 introduces *Senska*, the Enterprise Streaming Benchmark, including the design objectives, its architecture, the developed query list, and limitations. Section 4 concludes, giving an overview of Senska and illustrating areas for future work.

2 Related Work

As mentioned in [12], only few benchmarks for data stream processing architectures are available compared to the number of benchmarks for DBMSs. A brief comparison of selected benchmarks is shown in Table 2, which is based on the overview shown in [12].

The *Linear Road* Benchmark by Arasu et al. [7] is one of the most, if not the most popular application benchmark focussing on data stream processing. It includes a benchmarking toolkit comprising a data generator, a data sender as well as a result validator. With an execution of a benchmark implementation, a variable tolling system for a metropolitan area covering multiple expressways is simulated. The amount of accumulated tolls is dependent on multiple aspects of the traffic situation on these expressways.

The data sender emits the streaming data into the system under test (SUT). This input data contains four different record types, from which position reports are by far the most abundant records. The remaining data consist of three record types that express explicit user requests that always expect an answer from the system. Depending on the overall situation on highways, car position reports may require the SUT to create an output or not.

With regard to the benchmark workload, Linear Road defines four different queries with corresponding output types. For complexity reasons, the implementation of the lastly presented query was even skipped in the two implementations described in [7]. Besides streaming data, historical data covering ten weeks of tolling history is generated and partly has to be used in order to produce correct answers.

As a benchmark result, Linear Road defines one overall metric called L-Rating. The L-Rating indicates how many expressways a system can handle without violating the defined maximum response times for each query. The number of highways is a configurable parameter for the data generation step that is influencing the amount of input data.

The second benchmark presented in Table 2 is *StreamBench* [16]. It aims at benchmarking distributed DSPSs and 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 such 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 operators are to be compared, micro benchmarks have advantages over application benchmarks due to their simplicity. Measurements contain only the relevant parts without much overhead, which eases interpreting results.

StreamBench defines seven queries in total. They cover queries with single and multiple computational steps. Moreover, some queries require to keep a state in order to produce correct results while others do not. Only one query uses numerical data, while all others work on textual data. Overall, the seven queries cover a variety of functionalities, although some typical streaming operations like window functions are not taken into account.

Additionally, StreamBench defines four workload suites, which influence the way the benchmark is executed. The suite has an impact on, e.g., data scales, executed query set, the existence of an intentional node failure, or employed benchmark result metrics.

StreamBench makes use of two different real-world data sets. One of these contains textual data while the other one comprises numerical information. Generally, real-world data sets are always desirable as they represent real scenarios best and help increasing the benchmark's relevance. The two data sets used in StreamBench serve as seeds for data generation. Thus, synthetic data is used and reality is not represented entirely. Nevertheless, entirely using real-world data in a benchmark is certainly an ambitious aim as, e.g., scaling input data can easily become a challenge if the available data set is too small.

Contrary to Linear Road, StreamBench employs a message broker, which is used for decoupling data generation and consumption. This approach is similar to the benchmark architecture proposed in this paper, which is described in Section 3. In particular, Apache Kafka [14] is used as broker in StreamBench. Again, that is similar to the benchmark described in the present paper. A benchmark tool for data ingestion, such as the presented data sender that comes with Linear Road, is not described by the authors of StreamBench.

Table 2. Comparative Overview of Data Stream Processing System Benchmarks

	Linear Road	StreamBench	RIoTBench
Benchmark Type	Application	Micro	Micro and application
Considered SUT	DSPS or DBMS	Distributed DSPS	Distributed DSPS
Domain	Smart City (variable tolling)	Log processing and network traffic monitoring	Smart City, Smart Energy, Health (IoT in general)
Input Data	Synthetic (including historical data)	Synthetic (real-world data used as seed)	Synthetic (scaled real-world data sets)
Benchmark Result Metrics	One self-defined metric (throughput under latency restriction): L-Rating	Throughput or throughput-related, latency or latency-related, system availability	Latency, throughput, jitter (difference between expected and actual output rate), resource utilization
Query Result Validation	Validation tool provided	No tool provided	No tool provided

StreamBench defines different result metrics dependent on the workload suite. These include latency and throughput. The latter describes the average number of processed records per second and the amount of processed data in bytes per second. Both variants are calculated in total as well as per node. Moreover, three additional metrics are introduced: 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 benchmark tool.

The third benchmark shown in Table 2 is called *RIoTBench* [20] and focuses on benchmarking distributed DSPSs. It defines multiple micro benchmark scenarios as well as four application benchmark use cases, which represent combined micro benchmarks. These cover Extract, Transform and Load (ETL) processes, statistics generation, model training, and predictive analytics scenarios.

As input data, RIoTBench uses scaled real-world data sets from different IoT domains, namely, smart city, smart energy and health. A data sender tool for ingesting data into the SUT or an application for query result validation are not provided by the benchmark.

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

Summarizing, we see the need for a new application benchmark for assessing data stream architectures in an enterprise context for several reasons. First, currently only two major application benchmarks for data stream processing exist and only one of them considers characteristics of distributed systems in its metrics.

Second, historical data is not or only barely taken into account in all of the presented benchmarks. We believe that this is a crucial aspect in many enterprise contexts since, in order to achieve the greatest added value, streaming data needs to be combined with historical business data. As a consequence of that gap in benchmarks, 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.

3 Senska: Enterprise Streaming Benchmark

Due to the lack of satisfying ways to comprehensively compare stream processing architectures for enterprise scenarios, we aim to develop Senska, an enterprise streaming benchmark, which aims to fill this gap. In this section, the design objectives, the architecture, the query set, and limitations of Senska are illustrated.

3.1 Design Objectives

The design objectives beyond Senska follow the four criteria defined by Gray [10], namely relevance, portability, scalability and simplicity, which should be fulfilled by domain-specific benchmarks. These criteria are the basis for several publications that provide guidelines in the area of benchmark development, which illustrates their impact [9]. Although these aspects defined by Gray were already published in the early 90s, we still consider them valid for state-of-the-art benchmarks.

The chosen domain for Senska is industrial manufacturing since it represents a natural fit for an enterprise application requiring data stream processing capabilities, see Section 1.

Relevance The benchmark architecture, including data characteristics as well as the defined queries, will represent real-world scenarios and system environments as realistically as possible.

With respect to *data*, this ideally entails only employing real-world data in the benchmark. If no fitting data set can be found, synthetic data which is as close to real-world data as possible, will be generated. This can be achieved through generating data based on previously collected real-world data sets which alone would not be sufficient, similar to how, e.g., StreamBench tackles this issue, see Section 2.

Requirements in the context of input data exist, e.g., with regard to privacy concerns, size, or attribute variety. So in order to be a fitting data set, it should, e.g., be possible to publish the data as part of the benchmark, and its size as well as attribute variety should be appropriate in order to represent real-world environments.

Regarding data input rate, we aim to support multiple options in order to allow users to adapt Senska to their situation and be able to react to technology developments or environmental changes that might lead to increasing input rates in real-world environments.

Another area where relevance shall be considered are *queries*, i.e., the logic that is going to be executed by the system under test. This relevance in the sense of closeness to real-world scenarios shall be reached by validating the queries with industry partners from the corresponding benchmark domain, namely industrial manufacturing and similar industry sectors. This shall lead to a high acceptance of the benchmark and thus, to a higher credibility of its results as well as a higher motivation for implementing the benchmark. Although the benchmark focuses on a single domain, query characteristics, i.e., used functions, e.g., filtering or aggregating certain values of a data stream, are applicable to other areas and thus benchmark results can be beneficial for users from different domains. Moreover, Senska architecture and tools could be used for defining a new benchmark belonging to another domain, i.e., a query set with new data input that is be used for measurements.

Portability The benchmark definition shall be as much OS and system independent as possible. This enables implementing the benchmark for as many different environments as possible. Thus, a potentially large number of implementations can be reached, which helps gaining insights into a system's or environment's performance. Furthermore, a high number of implementations contributes to a high relevance and result credibility.

Senska aims to ensure portability by not restricting the choice of OS or the choice of used technologies for benchmark implementation. Although DSPSs might seem as a natural fit for data stream processing, it could potentially be exchanged with any other system or implementation that is able to answer the defined queries. With respect to the benchmark toolkit, compatibility with many OSs and platforms will be reached by using a JVM language.

Scalability The benchmark shall be able to handle smaller as well as bigger systems with regard to scale-up and scale-out architectures. Therefore, Senska will take the number of resources available into account when calculating benchmark results. Moreover, the provided tools, e.g., for data ingestion, are designed to handle scale-out architectures. This means no restrictions regarding number of nodes or CPU will exist in Senska and thus, there are no limits regarding scalability from the benchmark-side.

Simplicity The benchmark shall be simple to understand and to use in order to encourage people implementing it as well as to ensure credibility with respect to benchmark results. By providing a set of tools that are developed with the objective of increasing simplicity, people implementing Senska shall be supported as much as possible. This toolset comprises scripts for setting up the needed systems for data ingestion and validation, as well as applications for data ingestion, results validation, and benchmark metrics calculation. Additionally, an optional component for monitoring the environment will be included that can help identifying issues and getting an overview of key performance indicators.

3.2 Architecture

Figure 1 shows a high-level overview of our idea of an architecture for benchmarks with focus on data stream processing. Knowing about this simplified view helps getting an understanding of the idea beyond Senska's architecture. It shows three main components: the *data feeder*, *system under test* and the *result validator*.

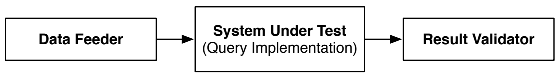

Fig. 1. General Architecture for Stream Processing Benchmarks in Fundamental Modeling Concepts (FMC)

The data feeder or sender is responsible for ingesting data into the SUT. The used term SUT is defined as "the system to be evaluated" [18]. In the context of Senska or stream processing benchmarks in general, the SUT processes incoming data and responds according to the defined queries. Produced results are ideally evaluated by a result validator in order to ensure correctness of the query implementations. This component could also calculate benchmark result metrics, e.g., latency or throughput. Besides calculating such metrics after running the benchmark, selected aspects might already be monitored when during each run in order to, e.g., react to failures or unintended behavior as soon as possible.

A more detailed overview of Senska's architecture is illustrated in Figure 2. All components are described in the following.

Input data Input data is represented through one or more files in CSV or similar format and represents sensor data from a manufacturing context. It acts as input for the system with respect to streaming data. In the best case, data will be entirely real-world data. If no suitable data set can be identified, synthetic data has to be used. Concretely, a data generation tool will be needed that takes care of creating the input data, preferably based on an existing real-world data set in order to keep characteristics. For the use within initial development of Senska, we use a data set from manufacturing context[1] published within the context of the DEBS Grand Challenge[2]. The challenge was conducted as part of the 2012 *Conference on Distributed Event-Based Systems*. In particular, the data set contains monitoring data, which was recorded by manufacturing equipment sensors. It includes about about 32.4 million records, which result in an uncompressed file size of about 5.5GB. As the end of the file can be reached quickly with high throughputs, we restart from the beginning when coming to that point in order to be able to run the benchmark for a certain period of time. That is an approach which is already proposed in, e.g., StreamBench [16].

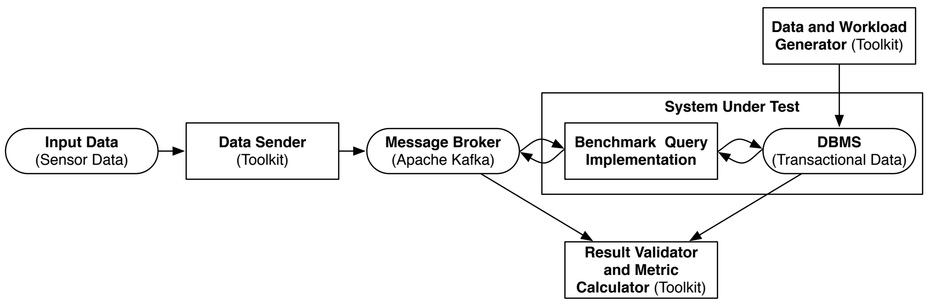
Data and Workload Generator (Toolkit)
System Under Test
Input Data (Sensor Data)
Data Sender (Toolkit)
Message Broker (Apache Kafka)
Benchmark Query Implementation
DBMS (Transactional Data)
Result Validator and Metric Calculator (Toolkit)

Fig. 2. Architecture of Senska in FMC

Data sender As part of the provided toolkit, the Senska data sender reads the input data and ingests it into the message broker component. The sending interval, i.e., the delay between sending consecutive records, is configurable. Although it is closer to reality to send records according to the timestamp they may contain, this might not be sufficient to satisfy configured data ingestion rates, i.e., to benchmark the SUT with certain data input rates. So it might not be possible to test, e.g., how much throughput an SUT can reach or how an SUT would behave with a doubled number of input records per second. Moreover, as reality and amounts of produced of data are rapidly changing, this flexibility allows staying up-to-date with a changing environment and prevents the benchmark from becoming outdated because of unrealistic data input rates.

[1] `ftp://ftp.mi.fu-berlin.de/pub/debs2012/`
[2] `http://www.csw.inf.fu-berlin.de/debs2012/grandchallenge.html`

The data sender, which is under development, is written in Scala and thus runs within a JVM on most common platforms. This compatibility with most OSs is the major reason for choosing Scala or a JVM language in general. As with all the tools in the toolkit, an open-source version of the data sender will be published with the first version of Senska.

Message broker The message broker represents a central part of Senska's architecture. It acts as interface between data sender and SUT. Furthermore, it is storage for query results in the defined setup. The message broker component in Senska is realized through Apache Kafka [14]. An overview about Kafka in the context of Senska is illustrated in Figure 3.

One reason for using Apache Kafka within Senska is its usage in enterprise software architectures. Among others, a common way of using Kafka is as interface to a DSPS. Thus, its role in Senska 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 [4] and Zalando [22].

An additional reason for using Apache Kafka is scalability with respect to ingesting data. If the data sender were to directly send data via sockets to the SUT, a change in the number of sockets would require changing the query implementations, since the additional connections would need to be handled by the SUT. Kafka *topics* provide a solution to this problem. An arbitrary number of *producers* can send data to a certain topic, which is internally distributed across the cluster and partitions by Kafka. The SUT application receives data from a topic, allowing the number of producers to be adapted. Using Kafka topics it is therefore possible to scale data ingestion in order to achieve a higher throughput without modifying query implementations. In order to ensure the correct order of records within Kafka topics, we use only one partition per topic, which can be seen in Figure 3.

Another reason for using Apache Kafka is latency measurements. In order to achieve latency results that are as correct and comparable as possible, we leverage Kafka's timestamp functionality. The timestamps before entering the SUT and after leaving the DSPS are taken into account for latency calculations. By doing so, it is possible to keep those calculations independent from the SUT and thus, no implementation modifications are needed and system-dependent differences or variations in terms of time measurements can be preempted.

Particularly, there are two types of timestamps Kafka offers to store with the messages, *create time* and *log append time* [2]. Create time is measured when a message is created and set by the Kafka producer that sends data to the cluster, and log append time is set by the broker when a message is appended to the Kafka log. In order to keep the measurements application-independent and so to avoid, e.g., unintentional application optimizations for reducing latency, we use log append time. Latency is computed by subtracting the output timestamp, i.e., log append time of the result record, from the input timestamp, i.e., log append time of the corresponding or last relevant input record. The downside of this approach is the included overhead time that is needed for transferring

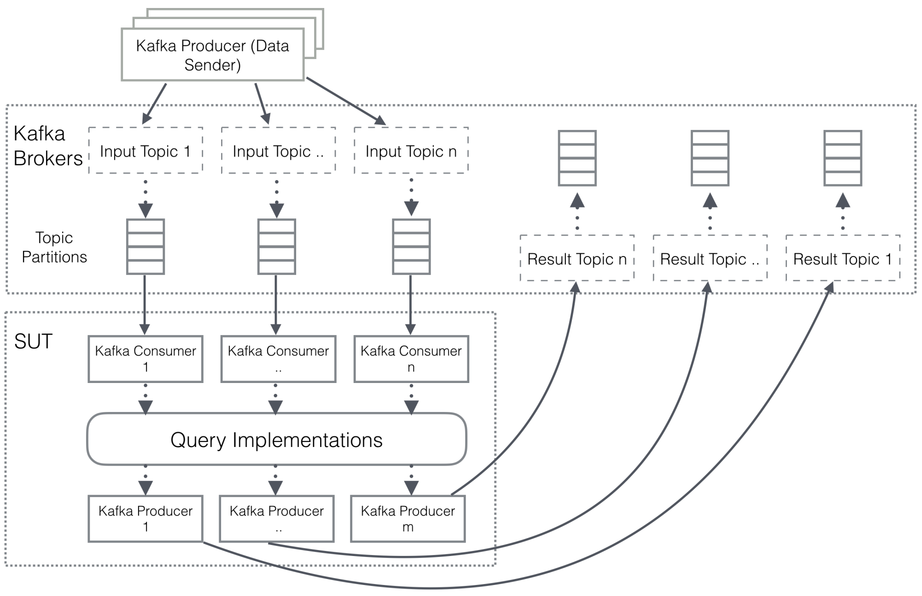

Fig. 3. Usage of Apache Kafka within Senska

messages from the broker to the SUT and back, which does not reflect the actual computation time of the SUT.

Although the exact processing times needed by the SUT are not measured by doing so, we do not consider this as an issue for the benchmark. Since all benchmarked systems follow the same approach, the overhead is included in all measurements and so results remain comparable in similar environments, i.e., if influencing parts, such as the network connection bandwith between Kafka and SUT, stay constant. Thus, we believe that the presented latency measurement approach is an objective way of calculating latencies in the context of Senska. It allows benchmarking any implementation that is able to retrieve and send data to and from Kafka. As times are taken independently outside of implementations, results are comparable and can not be unintentionally distorted by different time measurement mechanisms.

System under test The SUT comprises two main components: the benchmark query implementation and a DBMS. The queries defined by Senska and can be implemented using any technology, e.g., a DSPS or DBMS features such as stored procedures. The only requirement on the technology used for the implementation is the ability to communicate with a DBMS and with the message broker, i.e., with Apache Kafka. Historical or transactional data is consumed from a DBMS on demand, i.e., whenever a query requires this data. As the speed of the DBMS can influence the responsive time of queries, it belongs to the SUT. Some queries might require updating one or multiple historical data records, which is why a bidirectional connection between the query implementation and the DBMS exists. Query results are returned to the message broker, except for queries that require updating business data within the DBMS.

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

Result validator and metric calculator After finishing a benchmark run, the result validator and benchmark metrics calculator reads the query output from Kafka as well as from the DBMS and checks the correctness of results. This happens by rereading and reprocessing the input data from Kafka that was ingested into the SUT. Additionally, the benchmark results for the analyzed system or architecture, i.e., the scores for the benchmark metrics, are calculated.

When looking at Linear Road, the probably most noted benchmark of the three presented benchmarks in Section 2, the high-level architecture shown in Figure 1 is also valid. Senska as well as Linear Road have, next to the SUT, a data sender and a data validator component.

A look to Senska's architecture in Figure 2 reveals some differences, though. Linear Road also uses input data files that are sent by a data sender, but there is no message broker included. So the Linear Road data sender could directly send the input to the SUT. Furthermore, historical data in Linear Road are plain files that needs to be handled by the SUT. Thus, a DBMS is not required as in Senska. Contrary to Linear Road, there is a workload generator for historical data in Senska.

Similarly to Senska, a query result validator is provided as part of Linear Road. A data generator component is also included in Linear Road, which creates streaming and historical data. In Senska, there is a data generator for business data and, depending on the search of suitable real-world input data sets and as mentioned before, there might be a generator for streaming data, too.

3.3 Queries

When defining benchmark queries, relevance and simplicity need special consideration. The former one not only includes the closeness of queries to real-world scenarios, but also the coverage of important stream processing functionality.

For ensuring the latter one, we use the core set of operations for complex event processing (CEP) systems presented by [19] as a basis for functionality that should be covered by the queries. Although this list of operations is defined for CEP systems, it is applicable to data stream processing in general. Thus, we slightly modified to the original list to our needs and benchmark specifics, e.g., by adding the aspect of combining streaming with DBMS data.

1. Windowing
2. Transformation
3. Aggregation/Grouping
4. Merging (Union)
5. Filtering (Selection/Projection)
6. Sorting/Ranking
7. Correlation/Enrichment (Join)
8. Machine Learning
9. Combination with DBMS data

Table 3 shows an excerpt of the first queries defined in Senska. Next to the use cases behind each query, the covered functionalities are given, referencing the above stated list. Moreover, a query definition and a description are shown. Queries are defined similar to CQL continuous query language [6].

Table 3. Excerpt of Senska Query Set

#	Use Case	Tested Aspects	Query Definition	Description
1	Check Sensor Status	1;2;3	**SELECT** AVG(VALUE), MIN(VALUE), MAX(VALUE) **FROM** STREAM_1 **RANGE** 20 **SECONDS**	Calculate sensor statistics (avg, min, max) for, e.g., last 20sec.
2	Predict Sensor Values	1;8	**SELECT** PREDICTION(VALUE, 20 SECONDS) **FROM** STREAM_1 **RANGE** 300 **SECONDS**	Calculate expected values for, e.g., the next 20sec, for a sensor based on data of, e.g., last 300sec.
3	Identify Error I	1;5	**SELECT** COUNT(VALUE) > 5 **FROM** STREAM_1 **WHERE** VALUE > 40 **RANGE** 60 **SECONDS**	Log if sensor value exceeds defined limit for a certain number of times within a determined timeframe.
4	Identify Error II	1;5;7	**SELECT** * **FROM** STREAM_1 **AS** s1, STREAM_2 **AS** s2 **WHERE** s1.VALUE > 40 **AND** s2.VALUE < 10 **RANGE** 60 **SECONDS**	Log if two sensor values exceeds defined limit within a determined timeframe.
5	Check Machine Power	7;9	**SELECT** * **FROM** STREAM_1 **AS** s, DB_TABLE_1 **AS** t **WHERE** s.MACHINE_ID = t.MACHINE_ID **AND** s.POWER = 0 **AND** (s.TS > t.DOWNTIME_END **OR** s.TS < t.DOWNTIME_START)	Log if the machine is in an unscheduled phase of being turned off or in stand-by (assumption: there is always the next downtime stored in DB_TABLE_1).

Except for 4. Merging (Union) and 6. Sorting/Ranking, all functionalities of the presented operation list are covered. We aim to test these currently missing functions with the complete query set.

Currently, there is a first viable example benchmark implementation for the first query shown in Table 3, which only processes streaming data. For data processing, a DSPS is used. As shown in Figure 2, a data sender ingests the data into a Kafka cluster, which is the interface to the benchmark implementation. A result validator and metric calculator checks result correctness and computes latency for the queries.

3.4 Limitations

One limitation of Senska is its domain, namely industrial manufacturing. Senska, as every domain-specific benchmark, is focussed on a single application field, which can differ from other areas such as e-commerce software architectures. Differences may exist with respect to, e.g., data or query characteristics. Due to such varieties, there might be other benchmarks representing certain domain-specific circumstances better than Senska does. Nevertheless, queries defined by Senska cover functionalities, e.g., windowing or filtering, that can also be relevant to other domains.

Moreover, Senska only considers a limited data variety for simplicity reasons. To be more concrete, neither multimedia nor graph data is part of the benchmark and thus, capabilities of processing such data that a SUT might have will not be rewarded by Senska. So for scenarios where processing of such data is crucial Senska might not be the best choice of benchmark. Nevertheless, Senska will be open for extensions, which allows integrating such aspects.

A third limitation comes with the use of Apache Kafka as central part of Senska's architecture. As a consequence, a SUT must be able to retrieve and send data from and to Kafka. Since there are many clients available for Kafka that is not considered as a major drawback [1]. Besides, an architecture containing a message broker in general or Apache Kafka in particular might not be satisfying for everyone. Reasons can be, e.g., the wish to test another message broker or to test an architecture without message broker, which could mean retrieving streaming data within the SUT directly via one or multiple socket connection(s).

4 Conclusion

Within this paper, related work and the need for a new application benchmark for stream processing in an enterprise context is presented. The concept for such a new benchmark, namely Senska, is illustrated. Senska 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 transactional data in order to produce correct results.

By developing Senska, it is aimed to fill the gap that exists in the area of benchmarking enterprise architectures with focus on data stream processing. As a next step, we want to publish a first version of the benchmark together with an example implementation of the benchmark queries. Major future tasks are the search for a suitable input data set, and, if needed, the development of a data generator for scaling or generating streaming data. The example implementation has to be finished and the result validator adapted correspondingly. Additionally, the metric set as well as the tool responsible for its calculation have to be extended in order to cover all relevant aspects. Besides, the components related to business data have to be developed. Concurrently, the ideas shall be consistently be discussed with industry partners in order to validate design decisions with respect to relevance.

References

1. Apache kafka - clients. https://cwiki.apache.org/confluence/display/KAFKA/Clients, accessed: 2017-04-24
2. Documentation - kafka 0.10.2 documentation. https://kafka.apache.org/documentation/, accessed: 2017-04-24
3. Abadi, D.J., Carney, D., Çetintemel, U., Cherniack, M., Convey, C., Lee, S., Stonebraker, M., Tatbul, N., Zdonik, S.: Aurora: A new model and architecture for data stream management. The VLDB Journal 12(2), 120–139 (Aug 2003), http://dx.doi.org/10.1007/s00778-003-0095-z
4. Abdessemed, M.A.: 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
5. Arasu, A., Babcock, B., Babu, S., Datar, M., Ito, K., Nishizawa, I., Rosenstein, J., Widom, J.: Stream: The stanford stream data manager (demonstration description). In: Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data. pp. 665–665. SIGMOD '03, ACM, New York, NY, USA (2003), http://doi.acm.org/10.1145/872757.872854
6. Arasu, A., Babu, S., Widom, J.: The cql continuous query language: Semantic foundations and query execution. The VLDB Journal 15(2), 121–142 (Jun 2006), http://dx.doi.org/10.1007/s00778-004-0147-z
7. Arasu, A., Cherniack, M., Galvez, E., Maier, D., Maskey, A.S., Ryvkina, E., Stonebraker, M., Tibbetts, R.: Linear road: A stream data management benchmark. In: Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30. pp. 480–491. VLDB '04, VLDB Endowment (2004), http://dl.acm.org/citation.cfm?id=1316689.1316732
8. Dunning, T., Friedman, E.: Streaming Architecture: New Designs Using Apache Kafka and MapR Streams. O'Reilly Media (2016)
9. Folkerts, E., Alexandrov, A., Sachs, K., Iosup, A., Markl, V., Tosun, C.: Benchmarking in the cloud: What it should, can, and cannot be. In: Technology Conference on Performance Evaluation and Benchmarking. pp. 173–188. Springer (2012)
10. Gray, J.: The Benchmark Handbook - For Database and Transaction Processing Systems. The Morgan Kaufmann Series in Data Management Systems, Morgan Kaufmann (1993)
11. Hesse, G., Lorenz, M.: Conceptual survey on data stream processing systems. In: Proceedings of the 2015 IEEE 21st International Conference on Parallel and Distributed Systems (ICPADS). pp. 797–802. ICPADS '15, IEEE Computer Society, Washington, DC, USA (2015), http://dx.doi.org/10.1109/ICPADS.2015.106
12. Hesse, G., Matthies, C., Reissaus, B., Uflacker, M.: A new application benchmark for data stream processing architectures in an enterprise context: Doctoral symposium. In: Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems. pp. 359–362. DEBS '17, ACM, New York, NY, USA (2017), http://doi.acm.org/10.1145/3093742.3093902
13. Huber, M.F., Voigt, M., Ngomo, A.N.: 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. pp. 353–360 (2016), http://subs.emis.de/LNI/Proceedings/Proceedings259/article173.html

14. Kreps, J., Narkhede, N., Rao, J., et al.: Kafka: A distributed messaging system for log processing. In: SIGMOD Workshop on Networking Meets Databases (2011)
15. 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 2015 ACM SIGMOD International Conference on Management of Data. pp. 239–250. SIGMOD '15, ACM, New York, NY, USA (2015), http://doi.acm.org/10.1145/2723372.2742788
16. Lu, R., Wu, G., Xie, B., Hu, J.: Stream bench: Towards benchmarking modern distributed stream computing frameworks. In: Proceedings of the 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing. pp. 69–78. UCC '14, IEEE Computer Society, Washington, DC, USA (2014), http://dx.doi.org/10.1109/UCC.2014.15
17. Manyika, J., Chui, M., Bisson, P., Woetzel, J., Dobbs, R., Bughin, J., Aharon, D.: The internet of things: Mapping the value beyond the hype. http://www.mckinsey.com/~/media/McKinsey/Business%20Functions/McKinsey%20Digital/Our%20Insights/The%20Internet%20of%20Things%20The%20value%20of%20digitizing%20the%20physical%20world/The-Internet-of-things-Mapping-the-value-beyond-the-hype.ashx (June 2015), accessed: 2017-03-01
18. Menasce, D.A.: Tpc-w: a benchmark for e-commerce. IEEE Internet Computing 6(3), 83–87 (May 2002)
19. Mendes, M.R.N., Bizarro, P., Marques, P.: A Performance Study of Event Processing Systems, pp. 221–236. Springer Berlin Heidelberg, Berlin, Heidelberg (2009), https://doi.org/10.1007/978-3-642-10424-4_16
20. Shukla, A., Chaturvedi, S., Simmhan, Y.: Riotbench: A real-time iot benchmark for distributed stream processing platforms. CoRR abs/1701.08530 (2017), http://arxiv.org/abs/1701.08530
21. Southekal, P.H.: Data for Business Performance: The Goal-Question-Metric (GQM) Model to Transform Business Data into an Enterprise Asset (2017)
22. Vieru, M., López, J.: Flink in zalando's world of microservices. http://www.slideshare.net/ZalandoTech/flink-in-zalandos-world-of-microservices-62376341 (2016), accessed: 2017-04-06
23. Weiner, S., Line, D.: 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