StreamVizzard - An Interactive and Explorative Stream Processing Editor

Timo Räth
Technische Universität Ilmenau, Germany
timo.raeth@tu-ilmenau.de

Kai-Uwe Sattler
Technische Universität Ilmenau, Germany
kus@tu-ilmenau.de

ABSTRACT

Processing continuous data streams is one of the hot topics of our time. A major challenge is the formulation of a suitable and efficient stream processing pipeline. This process is complicated by long restart times after pipeline modifications and tight dependencies on the actual data to process. To approach these issues, we have developed StreamVizzard - an interactive and explorative stream processing editor to simplify the pipeline engineering process. Our system allows to visually configure, execute, and completely modify a pipeline during runtime without any delay. Furthermore, an adaptive visualizer automatically displays the operator's processed data and statistics in a comprehensible way and allows the user to explore the data and support his design decisions. After the pipeline has been finalized our system automatically optimizes the pipeline based on collected statistics and generates standalone runtime code for productive use at a targeted stream processing engine.

CCS CONCEPTS

• **Information systems** → **Data streaming**; • **Software and its engineering** → *Integrated and visual development environments.*

KEYWORDS

Stream Processing, Dataflow, Data Analysis, Data Visualization, Query Optimization, Code Generation

ACM Reference Format:
Timo Räth and Kai-Uwe Sattler. 2022. StreamVizzard - An Interactive and Explorative Stream Processing Editor. In *The 16th ACM International Conference on Distributed and Event-based Systems (DEBS '22), June 27–30, 2022, Copenhagen, Denmark.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3524860.3543283

1 INTRODUCTION

Working with continuous data in endless data streams is one of the big challenges of our time. Due to the steadily progressing digitization more and more applications require large and heterogeneous amounts of data that are produced, transmitted, stored, and analyzed. Strict real-time criteria often apply to the data, for example in critical sensor networks, so it is crucial to develop optimized and efficient stream processing applications to process the data in time.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
DEBS '22, June 27–30, 2022, Copenhagen, Denmark
© 2022 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9308-9/22/06.
https://doi.org/10.1145/3524860.3543283

This development process is complicated by two major issues. First, detailed knowledge of the data structure and characteristics is required to configure a stream processing pipeline that fulfills the requirements. This is especially important when optimizing and accelerating the operator execution time for example by utilizing modern hardware, like GPU or many-core CPU architectures. Second, many cycles of refinement are usually required until a suitable pipeline configuration has been found. This process involves repeated recompilation, deployment, execution, and evaluation of the pipeline program, before further adjustments and modifications can be applied, and significantly slows down the pipeline development.

Previous work aimed to solve these issues by developing advanced stream processing visualization and pipeline modification methods. [1] and [2] have shown that operators can be exchanged in running stream processing engines (SPE) like Apache Spark[1] and Apache Flink[2] without the need to restart however this requires a lot of additional work and long recompilation time. In [10] a system was developed to modify the data flow of a pipeline on runtime by utilizing control messages in Apache Flink. However, this approach can only be applied for predefined control flow variations. [4] and [8] have developed two tools to support the knowledge extraction from streaming data in an interactive way. However, all previous tools are either designed for static stream processing pipelines or require long recompilation times after a pipeline modification and therefore solve only some parts of the problem statement above.

In our demonstration, we present a new interactive and explorative stream processing editor - StreamVizzard - that aims to simplify and enhance the pipeline development process. By allowing completely modifying the pipeline structure on runtime the developer can immediately see the effects of his changes without long restart times. Together with our adaptive pipeline visualization the user can explore the processed data, understand the influences and effects of each operator and interactively develop a pipeline that fulfills his requirements.

In the next chapter, we will list requirements for such a system to solve the difficulties and issues in the pipeline development process as identified above. Afterward, we will present our system in Sect. 3 and describe how we implemented those requirements. In Sect. 4 we will demonstrate the easy extensibility of our system and give an outlook in Sect. 5, what visitors can experience at our demo.

2 REQUIREMENTS

Pipeline Configuration & Runtime Modification. A fundamental part of the pipeline engineering process is the configuration of the pipeline to execute. Usually, many iterations of execution, validation, and modification are required until a suitable pipeline

[1] https://spark.apache.org/
[2] https://flink.apache.org/

has been found. To speed up this slow process and remove long recompilation and redistribution times the system needs to allow to configure a pipeline and apply runtime modifications in real-time without any significant delay. Since the final structure and components of the pipeline are usually unknown at the beginning of the development process, operators, parameters, streams, and connections need to be fully exchangeable by such a system. This includes the addition and removal of operators as well as changes in the connections between operators.

Dynamic User-Defined Functions & Mode Join. User-defined functions (UDFs) are commonly used in any data science application and offer a powerful possibility to extend a stream processing application with custom functionality. During the development process, this functionality however might need to be changed depending on the other operators and the data flow of the pipeline. For example, a UDF that removes missing values from energy consumption data streams needs to be adapted completely when the data to be cleaned changes to recordings of a surveillance camera. To avoid a slow recompilation of the UDF the system needs to be able to understand those changes at runtime and immediately apply them so that the effect is visible without any delay.

Combining continuous data streams with existing machine learning models is a frequent use case for data science applications for example to predict changes based on sensor input data. During the development process, these models might need to be exchanged in case the input data or its characteristics changes. For example, an LSTM [9] model that is trained to predict the trend of a time series based on the last 100 sensor values collected by a preceding window needs to be exchanged in case the window interval increases to 200. The system needs to be able to dynamically exchange and reload existing models on the fly without the need for a restart.

Adaptive Data Visualization. The best pipeline configuration highly depends on the data that needs to be processed and requires deep insight into the characteristics and structure of the data. The system needs to be able to visualize the data that each operator processes and present valuable statistics like throughput or execution times to the developer in order to explore the data and support his pipeline design decisions. Since the operators, the data flow, and even the data sources might change completely during the development process, the system needs to automatically adapt its visualization methods to the changes in the data characteristics. Especially after the type of data changes an operator needs to immediately select a new suitable visualization method without any delay. For example, a window operator that visualizes incoming temperature sensor data with a time series diagram might be required to change its visualization to a word count histogram after the data changed to lines of text.

Automatic Pipeline Optimization. Optimizing the pipeline after the base functionality has been found is an important task to fulfill use case requirements like real-time processability of the incoming data. Besides some generic optimization methods described in [6] more adapted operator optimizations can be applied by utilizing knowledge about the data and operator statistics like data size or execution time. A system that aims to enhance and simplify the pipeline development process should be able to collect such statistics during the execution phase and automatically detect and apply suitable optimization methods where ever possible. Especially adaptations to the underlying hardware like executing an operator on the GPU can be done by the system to speed up the execution time of the pipeline.

3 SYSTEM DESCRIPTION

Based on the problem statement in Sect. 1 and the requirements in Sect. 2 we developed StreamVizzard - an interactive and explorative Stream Processing Editor to simplify the pipeline engineering process. StreamVizzard is a standalone system consisting of three major components which are illustrated in Fig. 1. First, a visual user interface to configure and modify a pipeline and explore the processed data and statistics. Second, an internal, dynamic stream processing engine, responsible for executing the pipeline and supporting runtime modifications. Third, a pipeline optimizer and code generator that automatically optimizes the pipeline based on collected data and statistics and produces adapted and efficient code for a target stream processing engine (SPE).

3.1 Adaptive Visualizer & User Interface

With existing data visualization tools like Apache Zeppelin[3] and Jupyter Notebook[4] and previous ideas like [3] and [7] in mind, we designed an adaptive and interactive user interface to modify the pipeline and explore the processed data. Based on the Javascript framework Retejs[5] we developed a web application that connects to our dynamic stream processing engine described in the next chapter over an API and allows to interactively configure and modify a pipeline. With a drag and drop approach, a user can choose from predefined or custom operators and intuitively connect them. To test the pipeline, the configured structure is sent to our engine, interpreted, executed, and results returned back over the API. The results which contain information about the processed data and operator statistics are then visualized by our web application. During the execution of the pipeline, the user can modify operator parameters, change connections between operators or remove and introduce completely new operators in real-time. All changes are continuously sent to the engine over the API to immediately apply them to the running pipeline. Since the processed data type of each operator might change during the execution we developed a generic and adaptive visualization component, that automatically adapts to changes in the data to visualize and chooses the best visualization method. Furthermore, valuable statistics are presented to the user in a comprehensible way like the throughput of each operator as well as processing speed and data volume to support the user in his pipeline design decisions. By presenting those statistics in form of a live heatmap, the user can detect bottlenecks or critical sections at a glance even for complex pipelines, and observe how these are influenced by changes in the structure or operator parameters.

3.2 Dynamic Stream Processing Engine

The internal dynamic SPE is the core component of StreamVizzard and is responsible for the flexible behavior of the system. Based on Apache Kafka[6] and the Python SPE Faust[7] we developed an adapted

[3] https://zeppelin.apache.org/
[4] https://jupyter.org/
[5] https://rete.js.org
[6] https://kafka.apache.org/
[7] https://faust.readthedocs.io/en/latest/

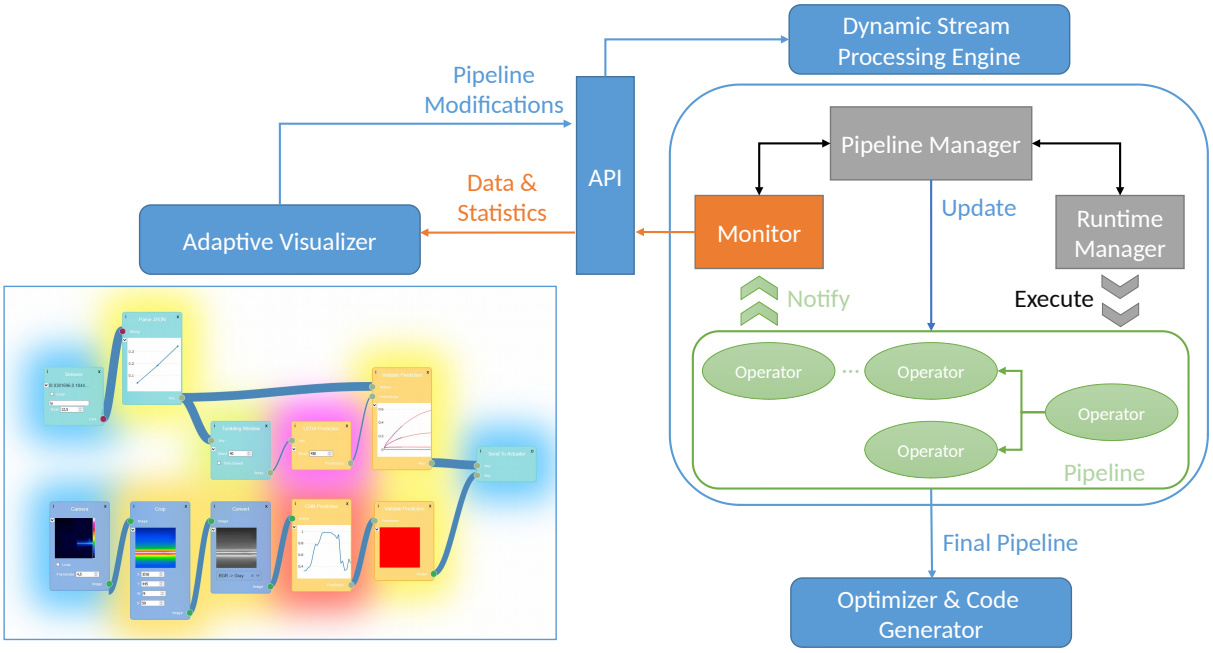

Figure 1: Architecture of StreamVizzard representing the different components and connections via the API. The Adaptive Visualizer demonstrates our web application to configure a pipeline and explore the processed data.

SPE with complete exchangeability of operators and connections in mind. During runtime connections between operators can be easily exchanged as well as new operators added or existing ones removed without any delay. This flexibility is possible due to the dynamic nature of each operator that checks after every processing step, which succeeding operators should receive its produced data tuple. A central pipeline manager coordinates the execution of operators and the exchange of data between the different components. When the structure of the pipeline is changed, for example through the web application, the pipeline manager triggers an internal update of the pipeline structure without the need to restart the pipeline. All operators keep their current state, even if parameters or connections changed, and continue executing after the update has been completed.

Using Python as an underlying programming language for our engine allows us to execute and exchange the code of UDFs during runtime without any delay. This allows the developer to immediately see and understand the effects of his changes and the influence on the pipeline procession.

As a tool designed and targeted for the pipeline development process, this dynamic nature of our engine comes at the cost of a slightly slower execution speed compared to a static engine, which is acceptable in exchange for the extensive flexibility in most cases. Furthermore, the pipeline optimizer described in Sect. 3.3 dissolves those dynamic components after the pipeline has been finalized to achieve the best execution speed for the targeted platform.

3.3 Automatic Optimizer & Code Generator

The pipeline optimizer and code generator is the final component of the system. After the development phase of the pipeline has been completed it should be optimized for productive use and exported to the target programming language and SPE. We have developed an optimizer, that is able to apply common stream processing optimization concepts described in [6] for example fusing subsequential operators to reduce communication overhead. Moreover that all dynamic components like the flexible relations between operators are removed to further speed up the execution time. During this process, the optimizer takes statistics like data types, throughput and data volume into consideration that were collected during the development phase to enhance the quality of optimization methods. When targeting GPU accelerated stream processing as described in [11] this allows to decide for efficient placement of operators depending on their processing time and data size. For example, an operator that performs a computation-heavy model join with a machine learning model might be placed onto the GPU while an operator that executes a simple preprocessing task is more efficient on the CPU.

The final task of this system component is the generation of standalone pipeline code for the targeted SPE. As a proof of concept, we have chosen the Python Faust SPE as a target platform to generate the code. As described in Sect. 4 this can be easily adapted to the desired SPE by utilizing our compilation interfaces.

4 EXTENSIBILITY

StreamVizzard was designed modularly with extensive adaptability in mind. The different components of the system are loosely connected over powerful interfaces and APIs as displayed in Fig. 1 and can be exchanged easily. For example, a custom visualizer can be implemented as a standalone monitoring application that communicates via defined APIs and socket connections with the core engine to present the state of the pipeline to the user. Moreover, instead of our visual drag and drop pipeline editor, other pipeline description approaches can be applied for example a custom Domain Specific Language (DSL) like SQL or a high-level query interface like in [5] to configure the pipeline to execute and modify.

```
class MyCustomOperator(Operator):

    def _execute(self, tupleIn: Tuple) -> Tuple:
        # Calculates the standard deviation
        # of the input data frame

        df = tupleIn.data[0]

        std = df.std()

        return self.createTuple(std,)
```

Listing 1: Demonstration on how to add custom operators to the stream processing engine.

StreamVizzard currently supports a variety of operators from different fields in the data science domain like stream processing, machine learning, and image processing. New operators can be easily added by extending our operator base class and configuring the functionality to execute. Everything else will be handled by our system. An example of the simple addition of new operators is presented in List. 1. Since our dynamic stream processing engine is executed in Python, all data science libraries are available to use for example Numpy[8], Pandas[9], OpenCV[10], and many more. This allows to quickly build an extensive database of custom operators for the required use case and execute them with our engine.

Analogously to the exchangeability of the visualizer a custom code generator can be added to compile the pipeline to the desired SPE and programming language. By utilizing our internal pipeline data structure the operator graph can be traversed and translated to the respective patterns of the target SPE. During this process collected statistics are available to improve the compilation decisions for example when targeting GPU computing. Besides generating code for common SPE like Apache Spark or Apache Flink also configurations for advanced analytical tools like AWS Kinesis[11] or Azure Stream Analytics[12] can be produced to utilize their analytical strengths on productive use.

5 DEMONSTRATION

The demonstration highlights the completely flexible and dynamic nature of our system and the benefits of using it to develop a stream processing pipeline. Visitors are invited to visually configure and execute a custom pipeline from a wide selection of operators in our web application and apply any modifications on runtime, like structural changes, to observe the instantaneous results in the execution. Guests who are familiar with Python as a programming language will also be able to define any UDF using different data science libraries and execute them on our system. We will provide different data sets that can be selected as input sources for the pipeline like sensor data and camera recordings. Also, the installed webcam can be used to receive live data and apply modifications to it.

By using our adaptive visualizer guests can explore the unknown data sets and experience the full pipeline engineering workflow from scratch. This involves identifying information about the different data types, their sizes, and their characteristics as well as detecting problematic sections or bottlenecks of the pipeline. To achieve this, operator statistics and our comprehensive heatmap can be utilized to get an at-a-glance overview of the system. By modifying operator parameters, and connections on runtime or even removing operators completely, users will be able to observe and understand the influence of their changes on the state of the pipeline.

[8] https://numpy.org/
[9] https://pandas.pydata.org/
[10] https://opencv.org/
[11] https://aws.amazon.com/de/kinesis/
[12] https://azure.microsoft.com/de-de/services/stream-analytics

ACKNOWLEDGMENTS

This work was partly founded by the Carl-Zeiss-Stiftung as part of the project 'Engineering for Smart Manufacturing' (E4SM).

REFERENCES

[1] Toon Albers, Elena Lazovik, Mostafa Hadadian Nejad Yousefi, and Alexander Lazovik. 2021. Adaptive On-the-Fly Changes in Distributed Processing Pipelines. *Frontiers in Big Data* 4 (2021). https://doi.org/10.3389/fdata.2021.666174

[2] Adrian Bartnik, Bonaventura Del Monte, Tilmann Rabl, and Volker Markl. 2019. On-the-fly Reconfiguration of Query Plans for Stateful Stream Processing Engines. In *Datenbanksysteme für Business, Technologie und Web (BTW 2019), 18. Fachtagung des GI-Fachbereichs „Datenbanken und Informationssysteme" (DBIS), 4.-8. März 2019, Rostock, Germany, Proceedings (LNI)*, Torsten Grust, Felix Naumann, Alexander Böhm, Wolfgang Lehner, Theo Härder, Erhard Rahm, Andreas Heuer, Meike Klettke, and Holger Meyer (Eds.), Vol. P-289. Gesellschaft für Informatik, Bonn, 127–146. https://doi.org/10.18420/btw2019-09

[3] George Chin, Mudita Singhal, Grant C. Nakamura, Vidhya Gurumoorthi, and Natalie Freeman-Cadoret. 2009. Visual Analysis of Dynamic Data Streams. *Information Visualization* 8 (2009), 212 – 229.

[4] Fabian Fischer, Florian Mansmann, and Daniel A. Keim. 2012. Real-time visual analytics for event data streams. In *SAC '12*.

[5] Philipp M. Grulich, Breß Sebastian, Steffen Zeuch, Jonas Traub, Janis von Bleichert, Zongxiong Chen, Tilmann Rabl, and Volker Markl. 2020. Grizzly: Efficient Stream Processing Through Adaptive Query Compilation. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data* (Portland, OR, USA) *(SIGMOD '20)*. Association for Computing Machinery, New York, NY, USA, 2487–2503. https://doi.org/10.1145/3318464.3389739

[6] Martin Hirzel, Robert Soulé, Scott Schneider, Buğra Gedik, and Robert Grimm. 2014. A Catalog of Stream Processing Optimizations. *ACM Comput. Surv.* 46, 4, Article 46 (mar 2014), 34 pages. https://doi.org/10.1145/2528412

[7] Stratos Idreos, Olga Papaemmanouil, and Surajit Chaudhuri. 2015. Overview of Data Exploration Techniques. *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data* (2015).

[8] Gonçalo Pires, Daniel Mendes, and Daniel Gonçalves. 2019. VisMillion: A novel interactive visualization technique for real-time big data. *2019 International Conference on Graphics and Interaction (ICGI)* (2019), 86–93.

[9] Ralf C. Staudemeyer and Eric Rothstein Morris. 2019. Understanding LSTM – a tutorial into Long Short-Term Memory Recurrent Neural Networks. https://doi.org/10.48550/ARXIV.1909.09586

[10] Jonas Traub, Nikolaas Steenbergen, Philipp Marian Grulich, Tilmann Rabl, and Volker Markl. 2017. I2: Interactive Real-Time Visualization for Streaming Data. In *EDBT*.

[11] Steffen Zeuch, Bonaventura Del Monte, Jeyhun Karimov, Clemens Lutz, Manuel Renz, Jonas Traub, Sebastian Breß, Tilmann Rabl, and Volker Markl. 2019. Analyzing Efficient Stream Processing on Modern Hardware. *Proc. VLDB Endow.* 12, 5 (jan 2019), 516–530. https://doi.org/10.14778/3303753.3303758