Interactive and Explorative Stream Processing

Timo Räth
supervised by Kai-Uwe Sattler
Technische Universität Ilmenau, Germany
timo.raeth@tu-ilmenau.de

ABSTRACT

Formulating a suitable stream processing pipeline for a particular use case is a complicated process that highly depends on the processed data and usually requires many cycles of refinement. By combining the advantages of visual data exploration with the concept of real-time modifiability of a stream processing pipeline we want to contribute an interactive approach that simplifies and enhances the process of pipeline engineering. As a proof of concept, a prototype has been developed that delivers promising results in various test use cases and allows to modify the parameters and structure of stream processing pipelines at a development stage in a matter of milliseconds. By utilizing collected data and statistics from this explorative intermediate stage we will automatically generate optimized runtime code for a standalone execution of the constructed pipeline.

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. 2022. Interactive and Explorative Stream Processing. 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/10.1145/3524860.3543287

1 INTRODUCTION

Data Science has become one of the hot topics of our time due to the continuously advancing digitization and networking of the world. Increasingly large and heterogeneous amounts of data are being produced and need to be transmitted, stored, and analyzed. Especially the real-time processing of endless data streams becomes more important to quickly react to changes, e.g. in sensor networks.

Numerous stream processing engines (SPE) have steadily evolved in recent years to cope with the ever-increasing data volumes by clever mechanisms such as distributed processing on large compute clusters (scale-out) or exploitation of modern hardware developments such as GPU computing and fast memory like PMEM (scale-up). However, despite all the technical tools and possibilities, the fundamental challenge still lies in the correct creation and configuration of the data processing pipeline to be able to draw the proper conclusions from the incoming data streams. This is complicated by two major problems that build on each other. First, the required pipeline structure highly depends on the data that needs to be processed, whose characteristics might not be clear in advance. Second, to evaluate if the configured pipeline fits the data and requirements, it needs to be compiled, distributed and executed before subsequent structural changes and adjustments can be made. This development process usually takes many iterations which not only requires a lot of time but also makes working with the data and understanding correlations much more difficult.

Two major challenges arise from this problem statement that we aim to overcome in our work. First, the smart and adaptive visualization of stream processing data to assist the process of data exploration, and second, the introduction of a dynamic pipeline structure that can be modified dynamically at runtime to simplify and speed up the engineering workflow.

Tools such as Apache Zeppelin[1] and Jupyter Notebook[2] have recognized this fundamental difficulty of gaining insight from data and have created ways to interactively visualize numerous datasets, making them easier to interpret. However, the applicability of such tools to data stream processing scenarios has proven to be limited due to its dynamic nature, especially for runtime changes of the pipeline structure or data characteristics.

Changes in the pipeline structure at runtime require a lot of additional work for current SPEs. [1] and [2] have shown that after such changes in Apache Spark[3] and Apache Flink[4], a costly recompilation and redistribution of the pipeline executable to all involved compute nodes is required to ensure a correct state transition. This does not only take a lot of time but also complicates the process of comprehending the influences of their changes for the developer. Previous work like [6] have already tackled this issue and developed interactive approaches to modify pipeline functionality at runtime however still limit the modification possibilities to a predefined extent.

Based on the presented challenges and the current state of the art, the overall goal of this Ph.D. thesis is to simplify and assist the pipeline engineering process. By proposing a new interactive and explorative approach to pipeline development we aim to transfer the strengths and advantages of visual knowledge extraction from datasets to endless and distributed stream processing applications. We will develop a dynamic pipeline structure that can be completely modified at runtime to instantly monitor and understand the effects

[1]https://zeppelin.apache.org/
[2]https://jupyter.org/
[3]https://spark.apache.org/
[4]https://flink.apache.org/

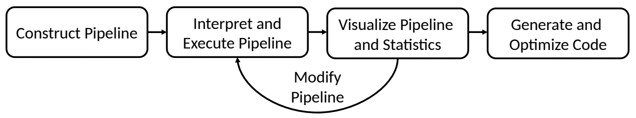

Construct Pipeline
Interpret and Execute Pipeline
Visualize Pipeline and Statistics
Generate and Optimize Code
Modify Pipeline

Figure 1: The conceptual workflow of the proposed approach

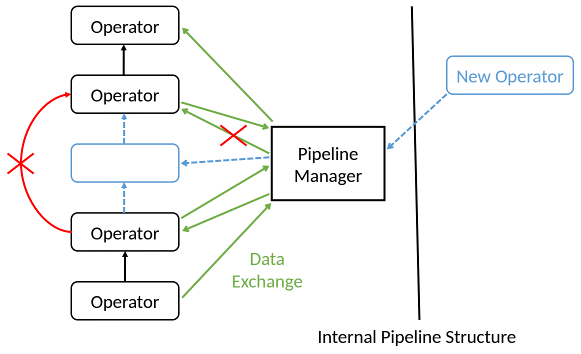


Figure 2: Internal pipeline representation with coordinating pipeline manager

of structural changes. Additionally, we will combine this with an automatic and adaptive visualization system that supports the extraction of data knowledge and characteristics and allows to follow the data transformations of each operator. Finally, we will provide an interface to compile the constructed pipeline in optimized and executable code utilizing all data and operator statistics that have been collected during the development process.

The remaining chapters of this paper will propose a proof of concept prototype for our thesis goal and describe the current state of the implementation.

2 PROPOSED SYSTEM

Based on the overview of the proposed workflow in Fig. 1 three consequential tasks arise from this that will be described in the subsections of this chapter.

- Implementation of a dynamic pipeline structure that can be modified at runtime [Sect. 2.1]
- Automatic and adaptive data visualization of the pipeline and its data [Sect. 2.2]
- Pipeline optimization and code generation utilizing collected data statistics [Sect. 2.3]

2.1 Dynamic Pipeline Structure

The core component of our proposal is the dynamic pipeline structure, that can be executed and modified at runtime without relevant delay. Since this kind of structural changes lead to long recompilation times with known SPEs, the approach we propose removes this limitation already in the core.

First, instead of converting the operator graph that represents the pipeline to execute into a static structure with fixed inputs, outputs, and connections, we will transform it into a flexible internal representation. This representation is displayed in Fig. 2 and consists of a pipeline manager, the operators of the pipeline, and their connections to each other. The pipeline manager is responsible for the pipeline execution and all internal communication and the core component of the system. After the pipeline is executed for the first time the manager will be started and running until the development process is completed. If a structural change occurs the pipeline manager will be notified and inform all affected operators about their new configuration and connections. This is possible because each operator works completely independently of the rest of the pipeline and checks in each processing step to which subsequent operators the produced data tuple should be passed. If a data tuple should be exchanged between two operators the manager will receive the respective tuple and a list of all recipients to distribute the tuple. This way operators can be easily added, removed, or reorganized by simply notifying the pipeline manager to trigger a reconfiguration without restarting the whole pipeline.

Second, we will not only remove long restarting times using this flexible structure but also the need for long code recompilation times by interpreting operator code instead of compiling it to a static version. This is possible by using an interpreted programming language like Python as the basis of our system and designing all operators accordingly in a flexible way to quickly adapt their functionality based on configurable parameters. Especially user defined-functions like maps or filters can benefit from this approach since the user code can be exchanged and modified on runtime in a matter of milliseconds without the need of operator recompilation. This will greatly simplify the pipeline development process and reduce the response time of the system which allows the developer to quickly test and experiment with different versions of their functions.

However, in addition to its compelling advantages, this approach also introduces a significant limitation. As with most interpreted programming languages, performance is usually noticeably behind that of compiled alternatives, since important optimization steps are missing. However, our system is primarily aimed at the development stage, where pure performance is usually not decisive. Therefore, this tradeoff between usability and performance can be accepted in favor of usability in most cases. After several change cycles, the pipeline may have delivered final satisfactory results and is ready to be transferred to a production-ready version. Here the limitations described can be softened by the code generator outlined in Sect. [2.3] in order to generate optimized, executable code.

2.2 Adaptive Pipeline Visualization

Tools such as Apache Zeppelin or Jupyter Notebook and [5] have shown that the visual representation of data and their correlations contribute significantly to comprehensibility and simplify an evaluation. Especially for stream processing scenarios, this assumption is true since the required pipeline structure highly depends on the data that needs to be processed. For this reason, detailed insight into the data structure and characteristics is required to develop an efficient and suitable pipeline architecture. Current visualization tools strongly rely on a clear definition of data types and visualization methods in advance. However, in a dynamic stream processing environment where the structure of the pipeline and therefore the processed data of each operator may change at runtime, these methods are not suitable anymore. For this reason, we propose an adaptive visualization mechanism that automatically chooses suitable visualization methods at runtime for each operator depending on its currently processed data.

This can be achieved by pre-registering supported data types like text, numbers, and images with corresponding visualization methods in our system. At runtime after each processed tuple operators will check if their produced data type has changed and select the next best visualization method if required. By using a modular design pattern our visualization components can be easily switched out at runtime without any significant delay.

Besides the actual representation of the processed data we will further support the pipeline engineering process by offering many more statistics that are important for structural design decisions. For example operator statistics such as throughput, data size, and processing time can be used to assess which operator could lead to bottlenecks or congestion and what influence this could have on the load on the transmission channels. These operator statistics are not only useful for the developer by supporting his design decisions but can also be utilized automatically by the pipeline optimizer and code generator to achieve more efficient pipeline code which is described in Sect. 2.3.

Finally, we will enhance the development process by providing a visual at-a-glance view of the overall state of the system. This will be achieved by representing the pipeline structure and statistics in a comprehensible way and offering different tools to keep track of important changes. This is especially important since the pipeline structure and processed data may change frequently during the development process. One major tool to visualize crucial aspects of the pipeline is the operator heatmap which will display the distribution of execution time, throughput, and memory consumption for each operator. By automatically adapting its scale to fit the current statistics distribution effects of structural changes like new bottlenecks, fully utilized operators, or congestion of communication channels can be recognized more easily.

2.3 Pipeline Optimizer and Code Generation

The pipeline optimizer and code generator forms the final component of our proposal. After the development process has been completed and a suitable pipeline has been found this component can be used to automatically optimize the pipeline based on the collected runtime statistics and generate executable standalone code. By converting the interactive and dynamic parts of the pipeline described in Sect. 2.1 into static pipeline components, the previous runtime constraints can be resolved and execution performance improved. Additionally, common stream processing optimization methods described in [4], such as operator reordering or operator fusing, can now be applied to further improve the pipeline structure and thus execution time. The target language in which the pipeline is to be generated is open and independent of our system. Through appropriate interfaces own code generators can be implemented to convert the given internal pipeline structure into the desired format. As a proof of concept we will implement our own optimizer and code generator that demonstrates the utilization of collected operator statistics to exploit data and runtime correlations.

Modern hardware developments such as GPU computing, faster memory, or many-core processors offer great potential and scope for further optimizing the execution and response times of data stream applications. In [7] it was shown, that speedups of multiple orders of magnitude can be achieved, by applying a clever and adapted mechanism utilizing the underlying hardware. We will focus on the efficient placement of operator functions, which can be executed either on the CPU or GPU since this decision can be greatly supported by the collected operator and data statistics. For example, information collected on the execution time and memory consumption of an operator in the processing pipeline could lead to the conclusion that this operator will most likely deliver better results on the GPU since the data transport to and from the GPU will be small compared to the computation time. Analogously, more statistics could be evaluated to assess the suitability of further optimization methods. As a result, we would like to achieve the best possible configuration depending on the pipeline structure, processed data, and execution environment.

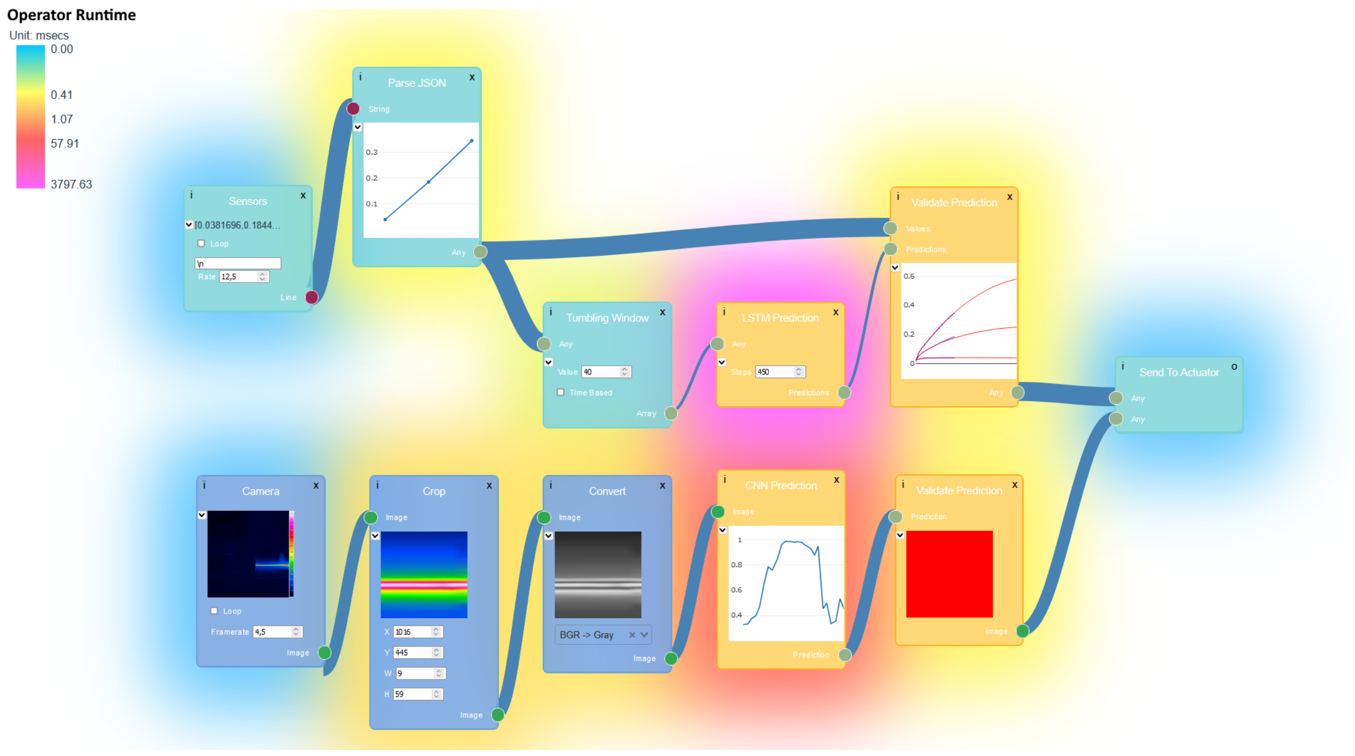
Additionally, we consider the GPU kernel fusing method to be particularly promising for our approach. As presented in [3] consecutive GPU kernels can be merged to reduce unnecessary back-and-forth data transport between CPU and GPU which is often still one of the bottlenecks for modern applications. Due to the nature of stream processing applications, this concept can be directly transferred to our code generator since for each operator that should execute on a GPU a separate kernel will be generated. By analyzing the processed data of each operator suitable kernels for fusing can be identified and merged to reduce data transfer and speed up the GPU utilization even more.

3 CURRENT WORK

The first tasks of our proposal have been completed and a prototype for real-time adaptability of a stream processing pipeline has been developed with promising results. Based on Apache Kafka[5] and the Python SPE Faust[6], a dynamic pipeline data structure was implemented which can be executed and modified at runtime in a matter of milliseconds. As a proof of concept, we decided on a scale-up approach on a single machine to demonstrate the feasibility of our prototype which can be migrated to a distributed system later on. To validate the functionality and correctness of the prototype numerous operators from the image processing, machine learning, and stream processing domain have been implemented and evaluated.

Furthermore, we have developed an adaptive visualization system using Rete.js[7] library that allows to easily construct and modify an operator graph based on the idea of flow-based programming. Fig. 3 demonstrates a sample pipeline that was constructed using this visual editor and is executed on our Python stream processing framework. Each operator displays its last produced data tuple by automatically detecting the best visualization method using our adaptive mechanism. Numerous data statistics are collected and

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



[0.0381696,0.1844...
\n
Rate 12,5
Value 40
Steps 450
Framerate 4,5
X 1016
Y 445
W 9
H 59
BGR -> Gray

Figure 3: Sample pipeline constructed with our visual editor and executed using our dynamic pipeline framework

presented to the user for example the throughput of the connections which is illustrated by using different types of line sizes. The figure also demonstrates one of our operator heatmaps, that visualizes the execution time of each operator and helps to detect bottlenecks of the system.

4 CONCLUSION

Formulating suitable stream processing pipelines for endless streams of incoming data is a difficult challenge and highly depends on the particular data to be processed. Many iterations of trial-and-error with long recompilation and restarting times complicate the development process. Additionally, in-depth knowledge of data characteristics and correlations is required to find an efficient solution.

By combining the advantages of visual and adaptive data exploration with the concept of real-time modifiability of a constructed pipeline at runtime we will add a highly interactive component to the so-far static process of pipeline creation. We have shown that our dynamic pipeline structure prototype already offers promising results as it is able to perform any modification to the pipeline structure or operator functionality in real-time which greatly simplifies and speedups the pipeline development process. Furthermore, our adaptive visualization approach is able to extensively support the knowledge discovery in endless data streams by automatically adapting to the currently processed data in real-time which is especially essential for dynamic pipeline structures. Finally, after a suitable pipeline has been found, our pipeline optimizer and code generator will resolve runtime and performance constraints due to the flexible nature of our pipeline and convert it to optimized and efficient standalone code by utilizing collected operator statistics.

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] Jiří Filipovič, Matúš Madzin, Jan Fousek, and Luděk Matyska. 2015. Optimizing CUDA code by kernel fusion: application on BLAS. *The Journal of Supercomputing* 71, 10 (July 2015), 3934–3957. https://doi.org/10.1007/s11227-015-1483-z
[4] 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 (March 2014), 34 pages. https://doi.org/10.1145/2528412
[5] Junyong In and Sangseok Lee. 2017. Statistical data presentation. *Korean J Anesthesiol* 70, 3 (May 2017), 267–276.
[6] Jonas Traub, Nikolaas Steenbergen, Philipp Marian Grulich, Tilmann Rabl, and Volker Markl. 2017. I2: Interactive Real-Time Visualization for Streaming Data. In *EDBT*.
[7] 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