Integrating event stream processing in ML and analytics systems

Jingyu Liu
Supervised by Vincenzo Gulisano
Department of Computer Science and Engineering, Chalmers University of Technology and University of Gothenburg
Gothenburg, Sweden
jingyu.liu@chalmers.se

1 Problem Statement

With the context of stream processing, this PhD project delves into strategies for efficiently integrating data analysis, model building, and serving within a cohesive and unified execution environment. The primary objective is to design a system for event stream processing-driven data analytics applications that fosters seamless interconnection and collaborative optimization, particularly for those that need to combine real-time stream processing with iterative analytics. Such integration is essential to address the growing demands of model-driven applications, which often require the ability to handle a wide spectrum of workloads. The proposed system aims to enhance operational flexibility, improve resource utilization, and maximize overall performance in data analytics workflows by supporting incremental stream processing and bulk-iterative tasks.

This comprehensive approach seeks to bridge gaps between traditionally distinct phases of data analysis, paving the way for a more dynamic and adaptive system. In simple terms, that is integrating stream processing and auto-tuning tools in the same analytic system to automate the relaxation trade-off aspects, such as applications' performance or computational costs.

2 Preliminaries

2.1 Stream processing

A data stream (e.g., the sequence of position reports from a moving car) is an unbounded sequence of tuples, sharing the same schema. Edge devices, such as the aforementioned vehicles, typically contain many embedded computers or sensors that function as edge devices, and these edge devices can support data pipelines which transform raw edge data into meaningful cloud-based insights within cloud-based systems. And these pipelines often rely on data-intensive processing paradigms like stream processing, where the applications are defined as Directed Acyclic Graphs (DAGs) of operators and run by Stream Processing Engines (SPEs), which control how these pipelines are deployed (through the distribution and parallelization of the operators in a DAG).

2.2 Reinforcement Learning

Reinforcement Learning (RL) is a foundational paradigm within the field of machine learning that concerns the problem of learning to make sequences of decisions through interaction with a dynamic environment. Formally grounded in the framework of Markov Decision Processes (MDPs) [9], RL enables an Agent to learn optimal behavior by receiving feedback in the form of scalar rewards, rather than explicit supervision. The Agent's objective is to reinforce itself to discover a policy, a mapping from states to actions, that maximizes the expected cumulative reward over time.

3 Related Work

Steam processing often faces the challenges of dynamically adapting to varying workloads, resource constraints, and performance requirements. RL, inspired by biological learning, with an Agent that interacts with an environment through trial and error to maximize rewards from its actions and determine the optimal policy [12], has been employed to automate diverse decision-making processes in a range of applications, including operator scheduling [2, 3, 5, 6], elasticity [1, 4, 10, 11, 13], and resource application [7, 8, 14], demonstrating the versatility of RL involving the unique challenge of stream processing environments.

3.1 Operator Scheduling

In Veith et al.'s papers [2, 3], RL is employed to model operator placement and reconfiguration tasks, automating decisions about where query operators should be deployed across the cloud-to-edge spectrum to optimize query execution by dynamically adapting to changing network conditions and resource availability.

Similarly, Huang et al. [5] leverage RL to address the task placement problem in heterogeneous stream processing environments, specifically using Apache Flink, to dynamically adjust the placement strategy for minimizing latency and maximizing resource utilization by modeling task placement as a decision-making problem, and Li et al. [6] propose a framework, guided by RL to simultaneously learn from limited runtime statistics and optimize the scheduling process, allowing the system to respond to real-time changes in workload patterns and resource availability with the help of jointly optimizing task scheduling and learning from operational feedback.

3.2 Elasticity

RL can also be applied in conjunction with elasticity control policies on heterogeneous resources to handle inherent uncertainties in system parameters [11]. This integration enables the system to dynamically adjust resource provisioning, ensuring efficient utilization while adapting to fluctuating workloads and maintaining operational stability. Building on these concepts, similar studies are discussed in [1, 10], where the authors focus on dynamically adapting application parallelism at runtime, emphasizing achieving a balance between meeting predefined QoS requirements and addressing the challenges posed by time-evolving workloads. Besides, the authors explore techniques to optimize resource allocation while minimizing energy consumption and operational costs, demonstrating the potential of RL to enhance system responsiveness and efficiency in dynamic environments.

Extending and building on these ideas, researchers applied RL within the Apache Spark SPE to guide auto-scaling decisions [13]. The proposed approach ensures that execution time constraints

are consistently met, even in the face of varying workload intensities. By leveraging RL, the system can predict future resource demands and scale computational resources accordingly, avoiding over-provisioning while maintaining performance standards. Further, traffic-aware scheduling mechanisms, such as the one presented in [4], introduce Q-learning to optimize task distribution across nodes in a heterogeneous cluster for both the workload characteristics and the available resources of each node, ensuring efficient utilization and minimizing bottlenecks.

3.3 Resource Allocation

From Ni et al.'s work [7], the proposed RL model exploits transfer learning in the context of resource allocation to apply knowledge from previous environments to new tasks, thus enabling the deep RL model to learn representations and generalize allocation policies across different stream processing scenarios by employing a graph-aware encoder-decoder framework. At the same time, Nie et al. [8] tackle the complexity of resource allocation in large-scale diverse stream processing graphs by using a coarsening strategy with the help of an RL-based approach.

4 Research Questions

The goal of stream processing technologies is to gain useful information from the massive and varied data stream. Since in this big data era, all of the applications are streaming data into SPEs, like Apache Flink, Spark Streaming, or Kafka Streams, and the system is racing to keep up with the volume that is increasing exponentially. However, optimizing such systems often involves making trade-offs among multiple performance metrics. For instance, improving throughput might come at the cost of reduced accuracy, or minimizing the latency might require increased resource consumption. It is not always the case that relaxing one metric (e.g., precision) necessarily increases latency; rather, the optimization space involves carefully balancing resource utilization, latency, accuracy, and throughput depending on the application's constraints and goals. The problem we want to solve is how to find a trade-off between the application's resource utilization and performance within a certain constraint, and to solve this, despite decades of research (see § 3), the dynamic adaptation of these methods is quite limited, and we believe using live-tuning tools, such as RL, can break those limitations.

5 Approach

In stream processing, operators are distinguished into stateless and stateful. Stateless ones do not maintain a state that evolves with the tuples they process, while stateful ones produce results from a state dependent on one or more tuples. Since the stream is unbounded, the state is limited to portions of time, usually called windows, maintained by W_{Size}– how long the duration for data captured in each window, and W_{Advn}– how often a new window starts, and users usually bind the scope of their analysis to windows containing the most recent and relevant data. Although many aspects of a pipeline can be optimized (e.g., operator placement, parallelism degree, or task scheduling), we target stateful operators, specifically Aggregates, to see how to find a trade-off between the application's resource utilization and performance, because the Aggregates are central to summarization and often serve as bottlenecks under workload fluctuations. To reach this trade-off, we consider using auto-tuning tools based on AI, and we think RL is one of the most mature and popular methods.

5.1 Memory compression through RL

Suppose there are two vehicles, one transmits data frequently, like once per second, resulting in constantly evolving data windows that are actively processed. In contrast, another vehicle sends data infrequently, maybe once per day. While the total volume of data from the second vehicle is small, the system still needs to maintain its window in memory, even though these windows are rarely updated or accessed, which leads to poorly used memory . In stream processing environments, especially those deployed at the edge or operating under resource constraints, memory usage is an important factor. Operators responsible for maintaining state across many such resources may encounter memory pressure when a large number of inactive windows accumulate over time, which motivates the need for on-demand memory management, particularly for rarely updated streams. To address this, our idea is to compress the windows for vehicles that send little data first, and decompress them later for outputting the results or shifting the window.

We can investigate an algorithm that uses a single parameter to control how much to compress, like a "knob", to control the amount of the window instances that should be compressed maintained by the Aggregate, within a definition of compression ratio, ranging from 0 – compressing all the windows, to 100 – no window is to be compressed. As for how to adjust this "knob", we introduce an Agent, trained by a neural network, that acts on the live Aggregate.

The components of the whole framework could be the following: (1) define an environment, mainly provided by the SPE; (2) find a controller, as the communication tool between the environment and the RL Agent; (3) construct RL Agent, with an appropriate training algorithm embedded inside the Agent. In the first place, the environment gives an initial state (such as input rate, throughput, output rate, latency, compression ratio, CPU consumption) to the Agent, which usually relies on a neural network. Secondly, the Agent gives a proper action (such as compress more, stay unchanged, compress less) to interact with the environment to update the state while generating rewards, and then sends it back to the Agent to reinforce itself to get as optimal actions as possible. This whole process looks like a continuous iteration until it reaches the terminal state or condition.

5.2 Window-adaption through RL

While SPEs offer flexibility in defining application logic through operator graphs, they often lack dynamic adaptability once deployed. Among the various operators, their performance and resource usage are directly influenced by how windows are defined and maintained. In most SPEs, window parameters, $W_{Advn}$and W_{Size}, are fixed at deployment time, which may result in suboptimal performance under dynamically changing workloads. For example, a large number of fine-grained windows (e.g., with small $W_{Advn}$and W_{Size}) can introduce considerable computational and memory pressure in periods of high input rate. Conversely, during low load periods, coarse-grained windows may lead to under-utilization of available resources and delayed output. This motivates us to explore adaptive window configuration as a means to balance the trade-off between performance

(e.g., latency and throughput) and resource utilization (e.g., memory and CPU).

We can propose a mechanism that allows the Aggregate to dynamically adjust its window configuration during runtime. Specifically, the Aggregate switches between a pre-defined set of (W_{Advn},W_{Size}) pairs in response to changes in the SPE state. By enabling such dynamic reconfiguration, the Aggregate will better align its behavior with the characteristics of the incoming data and the current system load.

Similar to the approach in 5.1, we introduce a tunable "knob" that determines how frequently the window configuration should be changed. For example, when the "knob" is set to a low value, the system is allowed to reconfigure more frequently, which may help it respond quickly to workload changes but at the cost of increased overhead. On the other hand, higher values result in less frequent changes, reducing reconfiguration cost but potentially missing opportunities for optimization. To determine this "knob", we can also use RL, in which the RL Agent observes states from the environment, selects actions, such as larger/smaller W_{Size}/W_{Advn}, or keep unchanged, to change the window configurations, and learns to adapt its policy over time based on the observed reward, probably defined in terms of system performance and resource consumption.

6 Evaluation Plan

We plan to evaluate our approach along several dimensions. First, we will assess the performance of the SPE introduced by the Agent that is activated to act on the live Aggregate through the changes in the different actions, using the stream processing usecases. We might also need to design and evaluate a complementary usecase characterized by significant variations in input rates and distributions to understand how our approach performs under more extreme and varied workload conditions. Beyond assessing the Agent's ability to support the relaxation of resource utilization and performance for the Aggregate under study, we will investigate the scalability of the approach.

7 Conclusions and Reflections

This PhD project is situated within the broader vision of building adaptive, intelligent stream processing systems that can respond effectively to dynamic data and workloads. In particular, as modern applications demand increasingly flexible and resource-efficient processing pipelines, it becomes crucial to enable automatic trade-offs between performance and resource usage.

In this work, we aim to take a focused step toward that goal by designing and evaluating an RL-based Agent that dynamically adjusts the behavior of a live Aggregate by applying a given number of actions while meeting a given latency threshold. We assessed our solution with several baselines, showing the Agent will support on-demand resource utilization for stream Aggregates.

Acknowledgments

Work supported by the Marie Skłodowska-Curie Doctoral Network project RELAX-DN, funded by the European Union under Horizon Europe 2021-2027 Framework Programme Grant Agreement number 101072456, Chalmers AoA Energy projects DEEP and INDEED, the Swedish Energy Agency (SESBC) project TANDEM, the Wallenberg AI, Autonomous Systems and Software Program and Wallenberg Initiative Materials for Sustainability project STRATIFIER.

References

[1] Valeria Cardellini, Francesco Lo Presti, Matteo Nardelli, and Gabriele Russo Russo. 2018. Auto-scaling in data stream processing applications: A model-based reinforcement learning approach. In *New Frontiers in Quantitative Methods in Informatics: 7th Workshop, InfQ 2017, Venice, Italy, December 4, 2017, Revised Selected Papers 7*. Springer, 97–110.

[2] Alexandre da Silva Veith, Marcos Dias de Assunçao, and Laurent Lefevre. 2019. Monte-carlo tree search and reinforcement learning for reconfiguring data stream processing on edge computing. In *2019 31st International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*. IEEE, 48–55.

[3] Alexandre da Silva Veith, Felipe Rodrigo De Souza, Marcos Dias de Assuncao, Laurent Lefèvre, and Julio Cesar Santos Dos Anjos. 2019. Multi-objective reinforcement learning for reconfiguring data stream analytics on edge computing. In *proceedings of the 48th international conference on parallel processing*. 1–10.

[4] Hamid Hadian, Mohammadreza Farrokh, Mohsen Sharifi, and Ali Jafari. 2023. An elastic and traffic-aware scheduler for distributed data stream processing in heterogeneous clusters. *The Journal of Supercomputing* 79, 1 (2023), 461–498.

[5] Xiao Huang, Yu Jiang, Hao Fan, Huayun Tang, Yiping Wang, Jin Jin, Hai Wan, and Xibin Zhao. 2021. TATA: Throughput-aware task placement in heterogeneous stream processing with deep reinforcement learning. In *2021 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)*. IEEE, 44–54.

[6] Teng Li, Zhiyuan Xu, Jian Tang, and Yanzhi Wang. 2018. Model-free control for distributed stream data processing using deep reinforcement learning. *arXiv preprint arXiv:1803.01016* (2018).

[7] Xiang Ni, Jing Li, Mo Yu, Wang Zhou, and Kun-Lung Wu. 2020. Generalizable resource allocation in stream processing via deep reinforcement learning. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 34. 857–864.

[8] Lanshun Nie, Yuqi Qiu, Fei Meng, Mo Yu, and Jing Li. 2023. Generalizable Reinforcement Learning-Based Coarsening Model for Resource Allocation over Large and Diverse Stream Processing Graphs. In *2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 435–445.

[9] Martin L Puterman. 1990. Markov decision processes. *Handbooks in operations research and management science 2* (1990), 331–434.

[10] Gabriele Russo Russo, Valeria Cardellini, and Francesco Lo Presti. 2019. Reinforcement learning based policies for elastic stream processing on heterogeneous resources. In *proceedings of the 13th ACM international conference on distributed and event-based systems*. 31–42.

[11] Gabriele Russo Russo, Matteo Nardelli, Valeria Cardellini, and Francesco Lo Presti. 2018. Multi-level elasticity for wide-area data streaming systems: a reinforcement learning approach. *Algorithms* 11, 9 (2018), 134.

[12] Richard S Sutton, Andrew G Barto, et al. 1998. *Reinforcement learning: An introduction*. Vol. 1. MIT press Cambridge.

[13] Kundjanasith Thonglek, Kohei Ichikawa, Chatchawal Sangkeettrakarn, and Apivadee Piyatumrong. 2021. Auto-scaling system in apache spark cluster using model-based deep reinforcement learning. *Heuristics for Optimization and Learning* (2021), 347–360.

[14] Zhan Zhang, Tianming Liu, Yanjun Shu, Siyuan Chen, and Xian Liu. 2023. Dynamic Adaptive Checkpoint Mechanism for Streaming Applications Based on Reinforcement Learning. In *2022 IEEE 28th International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, 538–545.