Holistic Adaptive Optimization Techniques for Distributed Data Streaming Systems

Victoria Vysotska[1], Iryna Kyrychenko[2], Vadym Demchuk[2] and Iryna Gruzdo[2]

[1] *Lviv Polytechnic National University, Stepan Bandera Street, 12, Lviv, 79013, Ukraine*
[2] *Kharkiv National University of Radio Electronics, Nauky Ave. 14, Kharkiv, 61166, Ukraine*

Abstract

This research article presents an approach to performance tuning in distributed data streaming systems through the development of the Holistic Adaptive Optimization Technique (HAOT). The importance of parameter tuning is underscored by its potential to significantly improve system performance without altering the existing design, thereby saving costs and avoiding the expenses associated with system redesign. However, traditional tuning methods often fall short by failing to optimize all components of the streaming architecture, leading to suboptimal performance. To address these shortcomings, our study introduces HAOT, a comprehensive optimization framework that dynamically integrates machine learning techniques to continuously analyze and adapt the configurations of sources, streaming engines, and sinks in real-time. This holistic approach not only aims to overcome the limitations of existing parameter tuning methods but also reduces the reliance on skilled engineers by automating the optimization process. Our results demonstrate the effectiveness of HAOT in enhancing the performance of distributed data streaming systems, thereby offering significant improvements over traditional tuning methods.

Keywords

Distributed Data Streaming Systems, Performance Tuning, Parameter Tuning, Machine Learning, Real-time Configuration Adaptation, Stream Processing, Automatic Parameter Tuning, System Optimization, Infrastructure Cost Savings, Data Pipeline Management, Evaluation Methods, Performance Metrics, Workflow Optimization, Data Management Techniques, Pipeline Efficiency, Process Improvement

1. Introduction

Our data-driven world demands immediate insights for quick decision-making. Real-time data processing offers the power to analyze information as it's generated. This is rapidly becoming essential in industries such as [1]:

- IoT Monitoring. In this realm, instant analysis of sensor data is transformative. It enables operational optimizations such as fine-tuning production processes in real-time, anticipating maintenance requirements before machinery fails, and triggering timely alerts to prevent incidents. This is not just about efficiency; it's about leveraging continuous streams of data to create a safer, more reliable, and cost-effective operational environment.
- Fraud Detection. In the financial and online retail sectors, real-time pattern recognition is a game-changer. By analyzing transactions as they happen, institutions can identify suspicious activities and halt fraudulent transactions before they are completed. This immediate response is crucial in a landscape where fraudsters continuously evolve their tactics. Beyond protection, this real-time vigilance enhances trust and security in digital platforms.
- Online Personalization. The digital consumer experience thrives on personalization. By analyzing user behavior data in real time, platforms can deliver tailored recommendations and content that resonate with individual preferences and current trends. This not only boosts

COLINS-2024: 8th International Conference on Computational Linguistics and Intelligent Systems, April 12–13, 2024, Lviv, Ukraine

victoria.a.vysotska@lpnu.ua (V. Vysotska); iryna.kyrychenko@nure.ua (I. Kyrychenko); vaddemgen@gmail.com (V. Demchuk); irina.gruzdo@nure.ua (I. Gruzdo)

0000-0001-6417-3689 (V. Vysotska); 0000-0002-7686-6439 (I. Kyrychenko); 0000-0003-3700-2344 (V. Demchuk); 0000-0002-4399-2367 (I. Gruzdo)

customer engagement but also drives sales and loyalty by providing a uniquely personalized shopping or browsing experience.

- Real-time processing overcomes the limitations of traditional batch analysis methods, offering key benefits [1, 2]:
- Improved customer experience. The ability to immediately respond to customer needs and preferences is invaluable. Real-time insights allow businesses to offer personalized experiences, resolve issues promptly, and thus significantly enhance customer satisfaction.
- Proactive Problem Solving. By identifying potential issues as they arise, businesses can address problems before they escalate. This proactive approach minimizes disruptions and can prevent minor issues from turning into major crises.
- Competitive Advantage. In today's fast-paced market, agility and responsiveness are critical. Businesses that can quickly adapt to changes and leverage real-time insights gain a crucial edge over competitors who rely on slower, more traditional data analysis methods.

As industries continue to evolve and generate more data, the demand for real-time data processing is set to increase. This trend underscores the shift towards more dynamic, data-driven decision-making processes that prioritize speed, accuracy, and efficiency. The future of business lies in the ability to harness the power of real-time data, transforming raw information into actionable insights instantaneously.

This article aims to provide a comprehensive guide to performance optimization techniques within streaming data systems. Aimed at practitioners with an intermediate understanding of streaming architectures, it will delve into common bottlenecks, practical optimization strategies, and essential monitoring tools. The article highlights optimization techniques applicable across significant streaming platforms while offering insights into platform-specific tuning methods.

2. Related Works

There is already a substantial amount of existing research focused on enhancing the performance of data streaming systems, underscoring the critical nature of this topic for various industries. These studies aim to address the complex challenges associated with processing large volumes of real-time data, striving to improve efficiency, reduce latency, and ensure reliability, which are pivotal for sectors such as finance, healthcare, manufacturing, and telecommunications. These extensive research papers reflect the growing recognition of the importance of data streaming performance in today's increasingly digital and data-driven business environments.

One of the most recent papers is research conducted by K.J. Matteussi, J.C.S. dos Anjos, V.R.Q. Leithardt, and C.F.R Geyer titled “Performance Evaluation Analysis of Spark Streaming Backpressure for Data-Intensive Pipelines” [3]. This research paper states that, unlike traditional datasets with finite boundaries, processing streaming data presents a unique set of complexities. Some of the most significant challenges include:

- Unbounded Data Streams. Streaming data sources continuously emit information, leading to a potentially infinite stream. Streaming systems must be able to process this constant influx without reaching storage or computational limits. This requires techniques like windowing to segment the data into manageable chunks and strategies to discard older data smartly.
- Low-Latency Requirements. Real-time insights are the bedrock of many streaming applications. Use cases like fraud detection or network monitoring demand swift action based on the most recent data. Streaming data systems must be designed to minimize delays in data ingestion, processing, and decision-making.

The authors delve into an in-depth analysis focusing on the performance aspects of backpressure mechanisms within Apache Spark Streaming environments, particularly in the context of data-intensive pipelines.

Another similar work titled “Beyond Analytics: The Evolution of Stream Processing Systems” [4] considers other challenges:

- Maintaining Accuracy and Consistency. Data streams often merge information from various sources, sometimes with differing formats and potential inconsistencies. Synchronization, error handling, and continuous data quality checks become essential for accurate, reliable results.
- Fault Tolerance. In a world of constant data flow, hardware failures or unexpected errors are bound to happen. Building in redundancy, distributed processing, and automated failover mechanisms minimizes the risk of losing data or experiencing service outages, which could be critical for business continuity in real-time systems.
- Scalability. The volume and velocity of data streams fluctuate. A spike in activity shouldn't crash the system. Services must seamlessly adjust their resources up or down in response to the shifting data flow to maintain performance and avoid costly over-provisioning.

The paper emphasizes a shift in how streaming systems are applied today. Instead of being confined to traditional tasks like window aggregates and joins, modern streaming systems are increasingly utilized for building scalable, general event-driven applications, which poses new challenges and considerations for their design, architecture, and intended use. This marks a significant shift in the stream processing paradigm, urging the database community to reevaluate current trends and methodologies.

Successfully optimizing streaming data systems relies on deeply understanding these inherent challenges and developing effective strategies to overcome them.

While the previous articles focus on the weaknesses of stream processing in various aspects, they highlight the importance of high availability and high performance. In the paper “A Survey on Automatic Parameter Tuning for Big Data Processing System” [5] the authors consider parameter tuning to address the common streaming problems. The researchers note that the significance of performance tuning is widely recognized in the industry, as appropriate configurations can lead to notable performance improvements without changing an existing system's design. In contrast, improper ones may result in significant performance declines. The benefits of parameter tuning techniques include cost savings on infrastructure and avoiding expenses associated with redesigning the system.

In recent years, considerable research has focused on automating performance tuning in distributed data streaming systems, employing a range of strategies: cost modeling, simulation-based, experiment-driven, machine learning, and automatic parameter tuning.

The work [6] considers the Cost Modeling approach for parameter tuning. It involves using cost models and statistics to find optimal parameter settings. A cost model estimates the resources (such as CPU, memory, and I/O) required for a given configuration of the streaming system. By understanding the relationship between system parameters and their impact on performance, developers can adjust settings to minimize costs while maintaining or improving performance. So, the approach is very efficient for predicting performance and has good accuracy. However, this approach requires a deep understanding of the system's internals and the ability to model the costs associated with different configurations accurately.

The simulation-based approach is considered in the work “CEPSim: Modelling and simulation of Complex Event Processing systems in cloud environments” [7]. Simulation-based tuning involves using a simulator to estimate the performance of an application under various settings. This approach allows for evaluating different configuration settings without affecting the actual production system. Simulations can model different aspects of the system, including network latency, processing speed, and data arrival rates, to identify the best configurations for specific workload patterns. The key advantages of this approach are the ability to test a wide range of scenarios and configurations without the need for extensive physical resources and efficiency in predicting fine-grained performance. The limitations of the simulation-based approach include complexity in fully replicating the intricate internal dynamics of a system, ineffectiveness in reflecting the fluctuating utilization of a cluster, and suboptimal efficiency in identifying the best settings.

The paper titled “Automatic Performance Tuning for Distributed Data Stream Processing System” [8] embraces the techniques mentioned above and considers other ones, such as experiment-driven, machine learning, and automation parameter tunning approaches.

The experiment-driven approach is a more hands-on method, where the application is executed iteratively with different settings to find the optimal configuration. This approach often involves a search algorithm that guides the selection of parameter settings based on the performance outcomes of previous experiments. Common search algorithms include grid search, random search, and more sophisticated ones like Bayesian optimization. This method is beneficial for complex systems where the relationship between parameters and performance is poorly understood and cannot be easily modeled. So, the approach works well for various system versions and hardware and finds the best performance for the real system, while it is time-consuming and cost-ineffective as it requires multiple runs.

Automatic tuning involves changing configurations dynamically while applications run based on real-time performance metrics. This adaptive approach allows the system to self-optimize in response to changing workloads and conditions. Automatic tuning requires continuous monitoring and a feedback loop where performance data is used to adjust the system's configuration immediately. This approach best suits environments with highly variable workloads and performance requirements.

The machine learning approach is well described in research papers [9] and [10] as well. It leverages data-driven techniques to model and predict application performance under different settings. By training a machine learning model on historical performance data and application logs, this approach can identify patterns and relationships that may not be apparent through traditional analysis. Once trained, the model can predict the performance outcomes of different configurations, allowing for rapid identification of optimal settings. This approach is particularly effective in dynamic environments where system behavior changes over time. However, this approach requires enormous datasets for training the model, which is very expensive to collect; otherwise, the accuracy is low [11, 12].

Beyond enhancing performance through parameter adjustments, a critical aspect is to determine performance metrics for streaming systems. The research papers [13, 14] consider the following performance metrics for streaming systems:

- Input Rate - this metric refers to the rate at which data is being received by stream processing applications. It is usually measured in records per second (or messages per second) and is crucial for understanding the volume of data being fed into the system. A high input rate compared to the system's processing capacity can lead to backpressure, where the system cannot process incoming data as fast as it arrives, potentially leading to increased latency or data loss.
- Processing Rate – this measures the rate at which the Spark Streaming application processes data, typically expressed in records per second. This metric should ideally be higher than the input rate to ensure that the system can keep up with the data flow without accumulating delays. The processing rate depends on various factors, including the complexity of the computations, the system's configuration, and the available resources (CPU, memory, etc.).
- Input Rows – this metric represents the total number of records received by the streaming application within a given time frame, often corresponding to the batch interval. Monitoring the number of input rows helps in understanding the data volume and ensuring that the system is configured correctly to handle the load. It also helps in identifying trends or spikes in the data input, which may require adjusting the system's capacity or optimizing the processing logic.
- Batch Duration – this is the time taken to process one batch of data and is a critical metric for understanding the performance of a streaming application. The batch duration includes the time to read the input data, process it, and write the output. The ideal batch duration should be less than the batch interval (the time between the start of consecutive batches) to

prevent the system from falling behind. If the batch duration is consistently longer than the batch interval, it can lead to increased latency and backpressure.

- Operation Duration - this metric, also known as the micro-batch duration, refers to the amount of time required to prepare and commit a micro-batch. This includes the time taken for tasks such as fetching the data from the source and writing the processed data to the sink. The operation duration is a subset of the batch duration and is crucial for diagnosing performance bottlenecks within individual stages of the data processing pipeline. A longer operation duration indicates that certain operations within the batch are time-consuming and may need optimization.

However, although Machine Learning and Automatic Tuning methods offer substantial advantages regarding resource optimization and simplicity in deployment, they mainly concentrate on enhancing the performance of streaming platforms like Spark Streaming, Flink, and Kafka Streams. This approach results in a critical shortfall: the overlooked improvement of data sources and destinations, which are essential elements of the streaming framework. Consequently, we have adopted the Holistic Adaptive Optimization Technique to overcome the constraints associated with conventional tuning approaches. This innovative strategy offers an extensive and flexible optimization framework, incorporating machine learning algorithms to perpetually evaluate and modify the settings of sources, streaming platforms, and destinations in a real-time manner.

3. Holistic Adaptive Optimization Technique for Parameter Tuning

In this section, we would like to consider the "Holistic Adaptive Optimization Technique" (HAOT) for parameter tuning in distributed stream processing systems. The HAOT is designed to address the limitations of traditional parameter tuning methods by providing a comprehensive and dynamic optimization framework that integrates machine learning techniques to continuously analyze and adapt the configurations of sources, streaming engines, and sinks in real time.

3.1. Methodology HAOT

HAOT operates on the principle of real-time performance feedback and adaptive configuration adjustments.

The approach involves the following key steps:

- Data collection continuously monitoring the performance metrics of the streaming application, including throughput, latency, and resource utilization, alongside the configurations of sources, streaming engines, and sinks.
- Utilizing machine learning algorithms to analyze the collected data and identify the relationships between the configurations of the sources, streaming engines, and sinks and their impact on application performance. This involves constructing a model that can predict the performance impact of varying configurations.
- Based on the insights gained from the dependency analysis, dynamically adjust the configurations of the sources, streaming engines, and sinks to optimize performance. This step employs machine learning models to predict optimal configurations under current operational conditions.
- As the streaming application runs, continuously update the machine learning models with new performance data, allowing the system to adapt to changing data patterns and operational conditions. This ensures that the optimization remains relevant and effective over time.

Fig. 1 represents the diagram of a system where HAOT is applied.

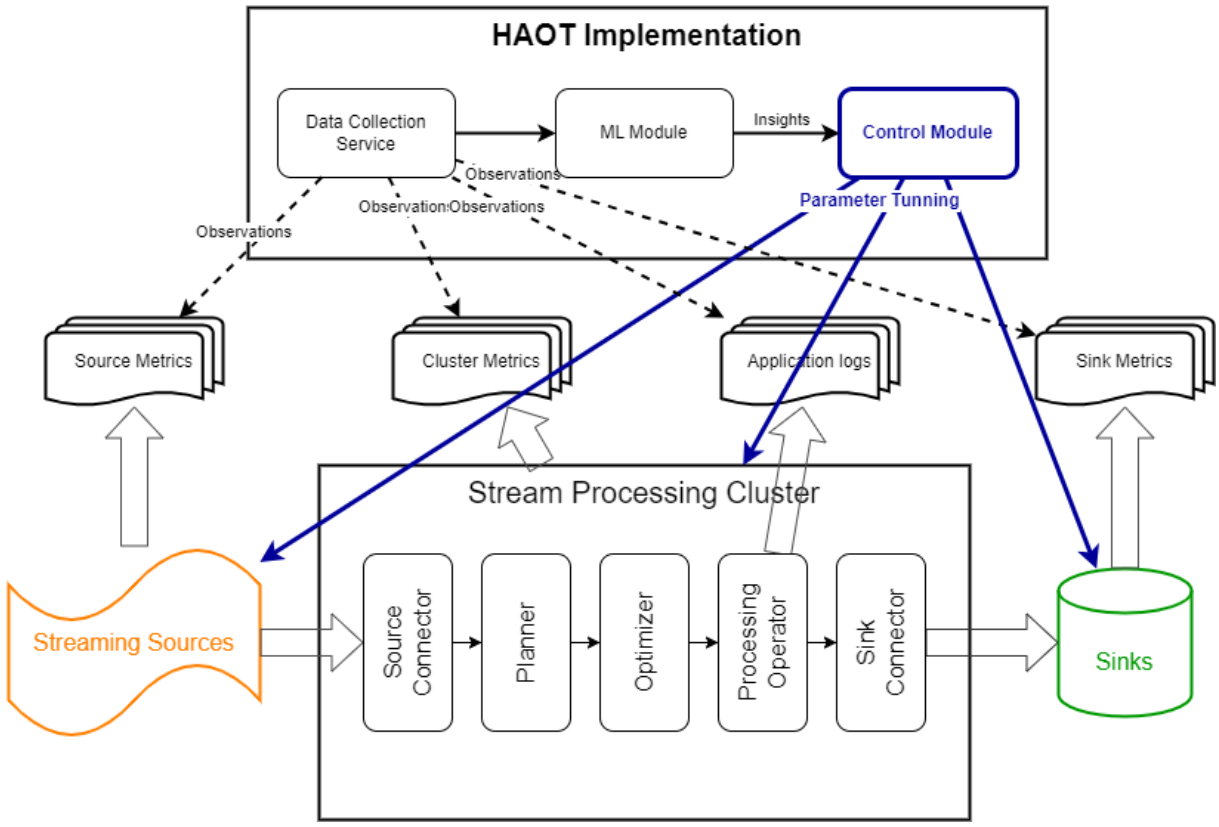

Figure 1: General architecture of a streaming system with applied HAOT

The architecture of distributed data streaming pipelines has the following components [1]:

- Streaming sources – these are the origins of data streams, which could be a variety of real-time data generating sources such as IoT devices, social media feeds, log files, or sensors. These sources continuously produce data that is ingested into the streaming pipeline. They are responsible for pushing data into the system in real-time or near-real-time;
- Streaming processing cluster – once the data is ingested from the streaming sources, it is handled by a streaming processing cluster. This cluster consists of distributed computing resources specifically designed to process large volumes of data in real-time. The processing cluster can perform a variety of tasks, such as filtering, aggregating, transforming, and analyzing the streaming data. It operates on data in a continuous fashion, often employing parallel processing and fault-tolerant techniques to handle high throughput and ensure data integrity. The cluster is scalable to accommodate varying loads of incoming data streams;
- Sinks (or outputs) – after the streaming data has been processed, it needs to be sent to a destination, known as a sink or output. Sinks can vary based on the requirements of the application and might include databases, data warehouses, file systems, or real-time dashboards. The choice of the sink depends on how the processed data will be used — whether for real-time analytics, stored for batch processing later, or used for immediate decision-making. The sink is the endpoint of the streaming pipeline where processed data is made available for use or further analysis.

HAOT introduces a few services into the existing architecture:

- Data Collection Service – the component that is responsible for gathering and organizing data from various sources that will be used for optimization using connectors. The data can include performance metrics, system logs, environmental conditions, and more, depending on the specific streaming pipeline. The Data Collection Service ensures that the data is collected in a consistent, efficient, and secure manner. It preprocesses the data (e.g., cleaning, normalizing, and transforming) to make it suitable for analysis and modeling. This service is crucial as the quality and relevance of the data directly impact the effectiveness of the optimization process.

- ML Model – is the key component of the proposed HAOT implementation. This model is designed to learn from the data collected and identify patterns, relationships, and trends that may not be apparent to human analysts. The ML model can be based on various algorithms, including regression, classification, clustering, or deep learning, depending on the problem at hand. It is trained with historical data to predict outcomes, optimize processes, or provide insights. Over time, the model can be retrained or adjusted as more data is collected and as conditions change, enabling it to adapt to new patterns and improve its accuracy and effectiveness.
- Control Module – acts as the decision-maker and executor within HAOT. It uses the insights and recommendations generated by the ML model to make informed decisions or adjustments to the system or process being optimized. This involves changing parameters or adjusting strategies. The Control Module is designed to ensure that these changes are made in a controlled, measurable, and reversible manner, allowing for continuous monitoring and adjustment based on performance feedback and evolving conditions. It serves as the interface between the analytical insights generated by the ML model and the operational components of the system, ensuring that the optimization is implemented effectively and efficiently.

3.2. Statement of Performance Tunning Problem for HAOT

The parameter tuning statement can be defined as finding optimal options using the arguments of the maxima (argmax) approach [5]. Let's consider a data streaming pipeline P executed on a stream data processing engine. It has the following form:

$$P = \langle j, s, t, r, c \rangle, \quad (1)$$

where j – a job executed as a part of the pipeline P;
s – properties of an input data stream;
t – properties of the target sink;
r – resources available for the cluster;
c – a configuration set of the source, job, and sink for the pipeline P.

Let c_i donate a parameter from the configuration set $c = \langle c_1, c_2, \dots, c_n \rangle$ that is a value from the finite domain $D(c_i)$, and n is the number of parameters in the set. Now we can define the configuration space S as the Cartesian product of the configuration domains $D(c_i)$:

$$S = D(c_1) \times D(c_2) \times \cdots \times D(c_n), \quad (2)$$

Given the pipeline job j that processes an input data stream s over cluster resources r and loads data to the sink t, the parameter tuning can be considered as evaluating the optimal configuration c_{opt} that maximizes performance metric function F over configuration space S:

$$c_{opt} = \underset{c \in S}{\operatorname{argmax}} F(j, s, t, r, c). \quad (3)$$

The primary distinction between the current method for determining the best parameters for a specific metric outlined in the paperwork [5] lies in considering the characteristics and settings of the input sources s and outputs – sinks r.

4. Experiment

To assess HAOT, we chose data streaming pipelines with the following technology stack:

- Sources: Confluent Kafka
- Processing Engine: Databricks Apache Spark
- Sink: Delta Lake on top of Azure Blob Storage

This pipeline was run daily for 2 months and had full logs and metrics. This allowed us to choose parameters using the ML model.

These technologies are characterized by a vast array of parameters, making it impractical at this stage to train the machine learning model with all possible variables. Nonetheless, the primary objective of this paper is to determine the feasibility of this approach. The parameters taken for the experiment are described in Table 1.

Table 1
Parameters of the Tested Distributed Data Streaming Pipeline

Technology	Component	Parameter	Description
Confluent Kafka	Source	partitions	A number of partitions of a Kafka topic. Ideally, this parameter should be adjusted by the control module. However, we will skip this at the current stage as this is a complex task [15].
Databricks Apache Spark	Processing Engine	maxOffsetsPerTrigger	Limit on the maximum count of offsets processed per triggering period
		minOffsetsPerTrigger	Limit on the minimum count of offsets processed per triggering period
		minPartitions	Preferred minimum quantity of partitions for reading from Kafka. By default, the Spark parallelism level is limited to the number of Kafka partitions. This parameter allows to increase number of parallel tasks.
		spark.sql.shuffle.partitions	Configures the number of partitions to use when shuffling data for joins or aggregations
		Trigger	The parameter determines the processing time interval of the streaming query
		spark.memory.fraction	Proportion of heap space allocated for execution and storage. A lower value results in more frequent occurrences of spills and evictions of cached data [16].
Delta Table	Sink	delta.targetFileSize	A fixed target file size for the table.
		delta.autoOptimize. autoCompact	This parameter controls the automatic compaction of small files in a Delta Lake table.
		delta.autoOptimize. optimizeWrite	This parameter governs the automatic optimization of the write operation into a Delta Lake table [17].

In the experiment, the following options were unchanged. The Kafka topic had 6 partitions and the target delta table had following options – autoCompact and optimizeWrite. The other options were set by the ML model.

The input dataset had the characteristics mentioned in Table 2.

Table 2
Input Dataset Characteristics

Property	Value
Size	80 million

Record Data Type	Avro using Confluent serializer
Average Record Size (In Kafka)	243 bytes – the value is low as the Avro format uses compact binary encoding to reduce the size of the serialized data
Total Dataset Size (In Kafka)	18 GB

The Databricks Spark cluster had 28 CPU cores and 184 GB of memory in total. The fine-grained characteristics are defined in Table 3.

Table 3
Databricks Spark Cluster

Cluster Component	Characteristic	Value
Driver	Node type	Standard_D4ads_v5
	CPU Cores	4
	Memory	16
Worker	Node type	Standard_DS13_v2
	CPU Cores	8
	Memory	56
	Spark Version	Spark 3.4.1, Scala 2.12
	Worker Count	3

The performance metrics were taken from Spark logs capturing Spark ProgressReporter [18]. Table 4 has the list of metrics and their correspondence with the parameters in the ProgressReporter. And additional metric was Kafka lag – the difference between current application offsets and maximum topic offsets [15].

Table 4
Databricks Spark Cluster

Metric	Source	Parameter
Input Rate	Spark Streaming	inputRowsPerSecond
Processing Rate	Spark Streaming	processedRowsPerSecond
Input Rows	Spark Streaming	numInputRows
Batch Duration	Spark Streaming	durationMs.triggerExecution
Operation Duration	Spark Streaming	durationMs
Kafka Lag	Kafka	metrics.avgOffsetsBehindLatest

The experiment setup:

- Base – the first part of the experiment involved running the Spark streaming pipeline with metrics collection and parameter tuning activated solely for Spark Streaming. This served as the baseline for performance metrics, where only the internal components of Spark Streaming were optimized based on available data, without external influences from other components of the pipeline.
- HAOT Applied – in the second part, the experiment extended metrics collection and parameter tuning to include not only Spark Streaming but also Kafka and the target Delta table (mentioned in table 1). This approach represents the holistic application of HAOT, where the optimization technique is applied across the entire data pipeline rather than in isolated segments.

5. Results

The ML model has set the following parameters for the first experiment – Base (see Table 5).

Table 5
Parameters of the run "Base"

Technology	Component	Parameter	Value
Confluent Kafka	Source	partitions	6 (Static)
Databricks Apache Spark	Processing Engine	maxOffsetsPerTrigger	240000 (default for the setup)
		minOffsetsPerTrigger	none (default)
		minPartitions	none (default)
		spark.sql.shuffle.partitions	50
		Trigger	AvailableNow (Static)
		spark.memory.fraction	0.7
Delta Table	Sink	delta.targetFileSize	none (default)
		delta.autoOptimize. autoCompact	true (static)
		delta.autoOptimize. optimizeWrite	true (static)

For the second experiment (HAOT Applied), the ML model provided different parameters, as mentioned in Table 6. While the option spark.memory.fraction has the same value, the other crucial options, such as minPartitions (the number of active tasks or Kafka consumers that will be created to read the topic), spark.sql.shuffle.partitions (the number of partitions after the shuffle), and delta.targetFileSize (the size of the target partitions) has different values.

Table 6
Parameters of the run "HAOT Applied"

Technology	Component	Parameter	Value
Confluent Kafka	Source	partitions	6 (Static)
Databricks Apache Spark	Processing Engine	maxOffsetsPerTrigger	960000 (default for the setup)
		minOffsetsPerTrigger	none (default)
		minPartitions	24
		spark.sql.shuffle.partitions	24
		Trigger	AvailableNow (Static)
		spark.memory.fraction	0.7
Delta Table	Sink	delta.targetFileSize	256 MB
		delta.autoOptimize. autoCompact	true (static)
		delta.autoOptimize. optimizeWrite	true (static)

The results of average metric values of the 2 runs ("Base" and "HAOT Applied") are presented in Table 7.

Table 7
Experiment Results

Metric	Description	The Base Run	HAOT Applied	Difference
Input Rate / sec	The bigger the better	29818	35843	20.21%
Processing Rate / sec	The bigger the better	30817	43335	40.62%
Input Rows	The bigger the better	239593	878511	266.67%

Batch Duration (ms)	The lower the better	16230	39515	143.47%
Operation Duration (ms)	The lower the better	7918	19637	148.00%
Total Execution Time (sec)	The lower the better	2835	1809	-36.19%

The bar chart of the data above is provided in Figure 2.

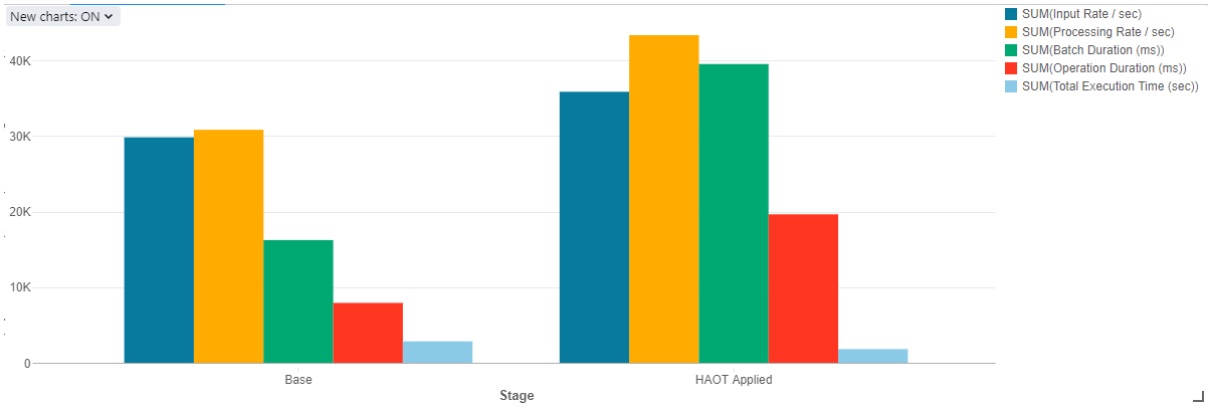

Figure 2: The bar chart represents the results of the 2 experiments

In the results above, we included the total execution time that was calculated as the difference in seconds between starting processing data and the moment all records were processed.

6. Discussion

Even though the Batch Duration and Operation Duration metrics went up for the HAOT case, the total execution time to handle the input dataset decreased by 36%. This means that, despite longer processing times for individual batches, the overall data loading speed was significantly improved. The reason for this rise is the latency to load bigger chunks of data from Kafka. Further supporting this improvement, the Input Rate, Processing Rate, and Input Rows metrics increased by 20%, 40%, and 267% respectively.

As mentioned above, the primary drawback of Cost Modeling, Simulation-based, and Experiment-driven approaches lies in their resource-intensive nature. These methods demand significant human and computational resources, making them costly to implement. Additionally, integrating these approaches into existing production systems poses substantial challenges due to their disruptive nature and the complexity of accurately modeling real-world scenarios.

On the other hand, Machine Learning and Adaptive Tuning approaches offer a more streamlined alternative, requiring minimal human intervention and not necessarily depending on highly specialized staff. These methods leverage algorithms to automatically adapt and optimize performance, reducing the need for continuous human oversight and detailed technical expertise. This aspect makes them particularly appealing to organizations seeking to optimize their streaming data systems with reduced operational overhead [19].

However, while Machine Learning and Adaptive Tuning strategies provide considerable benefits in terms of resource efficiency and ease of implementation, they primarily focus on optimizing the streaming engines themselves, such as Spark Streaming, Flink, and Kafka Streams. Neglecting the optimization of sources and sinks can lead to bottlenecks, data latency, and underutilization of the streaming engine's capabilities, ultimately impacting the overall performance of the streaming data system [20].

Therefore, while Machine Learning and Adaptive Tuning approaches streamline the optimization process and reduce the demand for skilled engineers, their effectiveness can be compromised if the optimization of sources and sinks is not adequately addressed. This oversight

can result in suboptimal system performance, highlighting the need for a holistic approach to optimization that encompasses all components of the streaming architecture.

7. Conclusions

The considered approach (HAOT) offers several advantages over traditional tuning methods:

- HAOT's methodical consideration of the entire data pipeline, encompassing the data sources, streaming engines, and sinks, ensures that optimizations are applied across the board rather than in isolated sections. This holistic view ensures that improvements in one area do not inadvertently impair performance in another, leading to enhancements in the overall system performance. By treating the data pipeline as an interconnected system, HAOT helps in identifying and rectifying bottlenecks or inefficiencies throughout, rather than just optimizing individual components in isolation.
- The ability of HAOT to adjust the configurations of the entire data pipeline in real time enables the system to react instantaneously to changes in data patterns, workload fluctuations, and variations in system conditions. This dynamic adaptability ensures sustained optimal performance under varying conditions without the need for manual intervention. As the system automatically adjusts to the current data environment, it maintains efficiency and effectiveness, preventing slowdowns or resource wastage.
- By continuously tuning configurations, HAOT minimizes the necessity for over-provisioning resources to handle peak loads, which is a common practice in traditional systems. Over-provisioning, while ensuring that the system can handle maximum expected loads, often results in underutilized resources during normal or low usage periods. HAOT's adaptive approach ensures that resources are allocated more efficiently based on current needs, leading to significant cost savings in infrastructure and operational expenses.
- Traditional system tuning often requires the expertise of highly specialized engineers, which can be a scarce and expensive resource. HAOT reduces this dependency by automating the optimization process, making it more accessible to a wider range of users.

While HAOT presents a promising approach to parameter tuning, there are challenges to be addressed, such as ensuring the scalability of the approach, dealing with the complexity of the machine learning models, and maintaining the balance between optimization frequency and system stability. Future work will focus on refining the approach to address these challenges, improving the efficiency of the machine learning algorithms, and extending the approach to cover a wider range of streaming platforms and application scenarios.

References

[1] P. Strengholt, Data Management at Scale: Modern Data Architecture with Data Mesh and Data Fabric, 2nd. ed., O'Reilly Media, Inc., 2023, pp. 173-175.

[2] H. Dulay, and S. Mooney, Streaming Data Mesh: A Model for Optimizing Real-Time Data Services, 1st. ed., O'Reilly Media, Inc., 2023, pp. 36-39.

[3] K.J. Matteussi, J.C.S. dos Anjos, V.R.Q. Leithardt, and C.F.R Geyer, Performance Evaluation Analysis of Spark Streaming Backpressure for Data-Intensive Pipelines, Sensors 2022. doi:10.20944/preprints202205.0334.v1.

[4] P. Carbone, and M. Fragkoulis et al., Beyond Analytics: The Evolution of Stream Processing Systems, SIGMOD, 2020.

[5] H. Herodotou, Y. Chen and J. Lu, A Survey on Automatic Parameter Tuning for Big Data Processing Systems, ACM Computing Surveys (CSUR), Vol. 53, 2020.

[6] M. T. Islam, S. Karunasekera, and R Buyya, Performance and Cost-Efficient Spark Job Scheduling Based on Deep Reinforcement Learning in Cloud Computing Environments, IEEE Transactions on Parallel and Distributed Systems, 2021.

[7] W. A. Higashino, M. A. M. Capretz, and L. F. Bittencourt, CEPSim: Modelling and simulation of Complex Event Processing systems in cloud environments, Future Generation Computer Systems, Vol. 65, 2016, pp. 119-140.
[8] H. Herodotou, L. Odysseos, Y. Chen, and J. Lu, Automatic Performance Tuning for Distributed Data Stream Processing System, IEEE 38th International Conference on Data Engineering (ICDE), 2022, doi:10.1109/ICDE53745.2022.00296.
[9] M. Trotter, T. Wood, and J. Hwang, Forecasting a Storm: Divining Optimal Configurations using Genetic Algorithms and Supervised Learning, IEEE International Conference on Autonomic Computing (ICAC), 2019. doi:10.1109/ICAC.2019.00025.
[10] H. Sagaama, N. B. Slimane, M. Marwani, and S. Skhiri, Automatic Parameter Tuning for Big Data Pipelines with Deep Reinforcement Learning, IEEE Symposium on Computers and Communications (ISCC), 2021. doi:10.1109/ISCC53001.2021.9631440.
[11] I. Gruzdo, I. Kyrychenko, G. Tereshchenko, O. Shanidze, Analysis of Models Usability Methods Used on Design Stage to Increase Site Optimization, 2023 7th International Conference on Computational Linguistics and Intelligent Systems (COLINS-2023), Vol. 3403, 2023, pp. 387– 409.
[12] I. Kyrychenko, I. Tereshchenko, G. Proniuk, N. Geseleva, Predicate Clustering Method and its Application in the System of Artificial Intelligence, 2023 7th International Conference on Computational Linguistics and Intelligent Systems (COLINS-2023), Vol. 3396, 2023, pp. 395– 406.
[13] G. V. Dongen, and D. V. D. Poel, A Performance Analysis of Fault Recovery in Stream Processing Frameworks, IEEE Access, Vol. 9, 2021. doi:10.1109/ACCESS.2021.3093208.
[14] K. J. Matteussi, J. C. S. dos Anjos, V. R. Q. Leithardt, and C. F. R. Geyer, Performance Evaluation Analysis of Spark Streaming Backpressure for Data-Intensive Pipelines, IEEE Access, Vol. 9, 2021, doi:10.1109/ACCESS.2021.3093208.
[15] E. Eldor, Kafka Troubleshooting in Production: Stabilizing Kafka Clusters in the Cloud and On-premises, Apress Berkeley, CA, 2023, pp. 25-36.
[16] A. A. Garcia, Hands-on Guide to Apache Spark 3: Build Scalable Computing Engines for Batch and Stream Data Processing, Apress Berkeley, CA, 2023, pp. 289-330.
[17] Phani Raj, Vinod Jaiswal, Azure Databricks Cookbook: Accelerate and scale real-time analytics solutions using the Apache Spark-based analytics service, Packt Publishing, 2021, pp. 240-302.
[18] B. R. Prasad, and S. Agarwal, Performance Analysis and Optimization of Spark Streaming Applications Through Effective Control Parameters Tuning, in: P. K. Sa, M. N. Sahoo, M. Murugappan (Ed.), Progress in Intelligent Computing Techniques: Theory, Practice, and Applications, Vol. 1, Springer Publishing Company, Incorporated, 2018, pp. 99-110.
[19] J. Nathan Kutz, Machine learning for parameter estimation, Proceedings of The National Academy of Sciences, Vol. 120, 2023, doi:10.1073/pnas.230099012.
[20] S. Nolan, A. Smerzi, and L. Pezze, A machine learning approach to Bayesian parameter estimation., npj Quantum Inf 7, Vol. 169, 2021. doi:10.1038/s41534-021-00497-w.