

Performance Benchmarking of Continuous Processing and Micro-Batch Modes in Spark Structured Streaming

Illia Fedorovych^{1,*†}, Halyna Osukhivska^{1,*†}, Nadiia Lutsyk^{1,†}

¹ Ternopil Ivan Puluj National Technical University, 56 Ruska St, Ternopil, UA46001, Ukraine

Abstract

This study investigates the performance of Continuous Processing mode in Apache Spark Structured Streaming, with a focus on its application in scenarios where low latency is a key requirement (such as traffic incident reporting or stock price prediction). Unlike the traditional micro-batching method, which processes data in intervals, the Continuous Processing mode allows for near-real-time data analysis by handling streams as they arrive. This approach significantly reduces latency, making it ideal for time-sensitive applications, but it does come with trade-offs in terms of throughput and fault tolerance. We conducted a comparative analysis of Continuous and micro-batching modes using various configurations and benchmarks, with a focus on latency and throughput metrics. Our findings indicate that while the Continuous Processing mode offers significantly lower latencies while using Rate source (2 ms instead of 528 ms in micro-batch mode), its performance in high-throughput scenarios using Kafka source may be less consistent (260 ms in contrast to 197 ms in micro-batch mode). The study also explores the practical implications of deploying Continuous Processing mode in real-world applications, assessing its compatibility with different data sources and sinks, predominantly Apache Kafka. These findings have practical implications for optimizing text data flow strategies in big data analytics, providing insights that can guide the selection of processing modes based on specific operational needs.

Keywords

Continuous Processing, Apache Spark, Structured Streaming, text data stream processing, big data, real-time processing, real-time analytics.

1. Introduction

Apache Spark 2.3 marked a significant evolution in the capabilities of Spark Structured Streaming by introducing a new mode known as Continuous Processing. Traditional Spark streaming had relied predominantly on micro-batching techniques, which process data in discrete intervals, combining streaming-like throughput with batch processing's fault tolerance and manageability benefits [1,2]. However, for scenarios demanding ultra-low

ITTAP'2024: 4th International Workshop on Information Technologies: Theoretical and Applied Problems, November 20–22, 2024, Ternopil, Ukraine, Opole, Poland


* Corresponding author.

† These authors contributed equally.

✉ illya.fedorovych@tntu.edu.ua (I. Fedorovych); osukhivska@tntu.edu.ua (H. Osukhivska);

lutsyk.nadiia@tntu.edu.ua (N. Lutsyk)

 0009-0003-7739-9689 (I. Fedorovych); 0000-0003-0132-1378 (H. Osukhivska); 0000-0002-0361-6471 (N. Lutsyk)

 © 2024 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

latency, micro-batching could introduce inherent delays incompatible with time-critical applications, such as traffic anomalies detection.

Continuous Processing in Apache Spark Structured Streaming represents a paradigm shift from the traditional micro-batching execution strategy. While micro-batching processes data in discrete chunks, Continuous Processing treats streams as proper continuous flows, reducing latency substantially. This is achieved by utilizing long-lived operators that maintain their state across data batches rather than resetting for each batch as in micro-batching. This approach minimizes the overhead caused by frequent stopping and starting task executions, which is typical in micro-batch processing.

The execution strategy of Continuous Processing is designed around the concept of pipelines akin to traditional database streaming models. These pipelines utilize long-lived tasks that continuously receive and process data as it arrives without the predefined intervals that characterize micro-batching. This model allows Continuous Processing to deliver lower latency by reducing the delays associated with batching.

As of the latest release of Apache Spark, version 3.5.1, Continuous Processing is still labeled as experimental [3]. This designation implies that while the functionality is available for use, it is not yet fully optimized and may lack some robustness features integral to micro-batching. Furthermore, the integration of Continuous Processing with various data sources and sinks is currently limited to Apache Kafka, Console Sink, and Rate Source.

The limited support for diverse sources and sinks means that deploying Continuous Processing in a production environment requires careful consideration of the input and output interfaces. For organizations heavily using Kafka, Continuous Processing offers a promising opportunity to enhance the responsiveness of their streaming applications. However, for use cases involving other sources or more complex transactional needs, the experimental nature and the current limitations might pose significant challenges.

The ongoing development and incremental improvements in Continuous Processing suggest that future releases of Spark may expand its capabilities and integrations. Developers and organizations interested in cutting-edge streaming technologies should monitor updates to Spark's Continuous Processing mode, as it will likely evolve significantly.

This paper analyzes these two processing modes within Apache Spark Structured Streaming, focusing on their performance implications across various setups and deployments. By examining several workloads, this study aims to delineate the conditions under which Continuous Processing significantly outperforms micro-batching and, conversely, when micro-batching remains preferable.

2. Related Works

Armbrust et al. [4] provided a general analysis of Continuous Processing mode and researched the relationship between latency and throughput. The paper also provided information about the scalability of micro-batching mode, showing Structured Streaming's throughput depending on the number of worker nodes. However, despite discussing the architectural difference between micro-batching and Continuous Processing modes, they were not directly compared regarding latency and throughput given the same resources.

Other works primarily focus on using micro-batching mode [5] or comparing it with other data stream processing frameworks, such as Kafka Streams, Storm, Flink, etc. [6,7]. Van

Dongen and Van Den Poel conducted the most extensive benchmarking, presenting an open-source benchmark implementation that can measure latency and throughput under different workloads, such as sustainable and bursts [8].

In the paper [8], the Continuous Processing mode was mentioned a few times as a lower-latency alternative to micro-batching, but it was discarded because of its experimental status. The authors later expanded the list of measured metrics by publishing papers on the scalability and fault tolerance of data stream processing frameworks [9,10].

3. Proposed methodology

The proposed methodology uses micro-batching and Continuous Processing to compare latency and throughput under different configurations. The benchmarking process uses a pipeline with either a rate source or a Kafka source as input, transforming the timestamp and value into a JSON string, and a Kafka sink as the output.

Key components of the method include input sources, which involve testing the rate source at 10 and 100 records per second (rps) and using a Kafka source that utilizes text data from a Ukrainian Wikipedia corpus. The output sink is configured with a Kafka sink, which maps the message’s timestamp into the payload as a “new_timestamp” field to measure each record’s latency. Spark configuration tests are conducted in local mode with 3 and 16 workers, and each benchmark is executed for 2 minutes. Additionally, the number of partitions for Kafka topics (both source and sink) is adjusted to evaluate its impact on throughput and latency.

The proposed methodology aims to identify the optimal configuration for both micro-batching and Continuous Processing modes by systematically varying the number of records per second, worker nodes, and Kafka partitions. This approach helps understand each processing mode’s limitations and performance characteristics under different loads. The final configurations are presented in Table 1.

Table 1
Initial benchmarking configurations

Source	Workers	Mode	Sink
Rate (10 rps)	3	Micro-batch	Kafka
Rate (10 rps)	3	Continuous	Kafka
Rate (100 rps)	16	Micro-batch	Kafka
Rate (100 rps)	16	Continuous	Kafka
Kafka	3	Micro-batch	Kafka
Kafka	3	Continuous	Kafka
Kafka	16	Micro-batch	Kafka
Kafka	16	Continuous	Kafka

After running the initial benchmarking and analyzing the results, it was suggested that the default number of partitions in the Kafka topic might influence Spark’s throughput and latency. It was decided to include parametrization of this number to exclude the possibility of Kafka being the bottleneck in the benchmarking process. The configurations for benchmarking different Kafka topic partition count are presented in Table 2.

Table 2

Kafka topic partition count benchmarking configurations

Source	Kafka partitions	Workers	Mode
Kafka	3	3	Continuous
Kafka	16	16	Continuous
Kafka	48	16	Continuous

To develop an understanding of at what rates continuous processing becomes unreliable, another set of benchmarks focused on how continuous and micro-batch processing handle different loads in test conditions, which could provide a controlled throughput rate, was decided to perform.

The resulting configurations for performing sustainable throughput benchmarks included comparing micro-batching and continuous modes, using rate source at 10 - 10 000 rows per second, 3 and 16 worker nodes, and writing into Kafka topics with 3 - 32 partitions, with the number of Spark partitions equal to Kafka's.

The total number of configurations was 64, but some configuration executions were canceled during the benchmarking process to make readjustments, as clear patterns have emerged, rendering some of the unperformed configurations worthless. The list of configurations in the stage 2 benchmark is presented in Table 3.

Table 3

Sustainable throughput benchmarking configurations

Sources	Workers	Partitions	Mode	Status
Rate (10 rps)	3/16	3/10/16/32	micro-batch/continuous	Performed
Rate (100 rps)	3/16	3/10/16/32	micro-batch/continuous	Performed
Rate (1 000 rps)	3/16	3/10	micro-batch/continuous	Performed
Rate (1 000 rps)	3/16	16/32	micro-batch/continuous	Canceled
Rate (10 000 rps)	3/16	3/10/16/32	micro-batch/continuous	Canceled

After observing the results of sustainable throughput benchmarks, it was decided to expand the research by decoupling the Spark partition count from the Kafka partition count to see if further improvements could be made. The benchmarking configurations are provided in Table 4.

Table 4

Decoupled Spark/Kafka partition count benchmarking configurations

Sources	Spark Partitions	Kafka Partitions	Mode
Rate (1 000 rps)	32	16	micro-batch
Rate (1 000 rps)	64	16	micro-batch
Rate (1 000 rps)	16	16	micro-batch
Rate (1 000 rps)	16	32	micro-batch
Rate (1 000 rps)	16	64	micro-batch

Rate (10 000 rps)	16	16	micro-batch
Rate (10 000 rps)	16	32	micro-batch
Rate (10 000 rps)	16	64	micro-batch
Rate (1 000 rps)	3	16	continuous
Rate (1 000 rps)	10	16	continuous
Rate (1 000 rps)	16	3	continuous
Rate (1 000 rps)	16	10	continuous
Rate (1 000 rps)	16	16	continuous
Rate (10 000 rps)	16	3	continuous
Rate (10 000 rps)	16	10	continuous
Rate (10 000 rps)	16	16	continuous

4. Results

After setting up and running initial benchmarks on modern versions of Kafka and Spark (Kafka 3.7.0, released February 27, 2024; and Spark 3.5.1, released February 23, 2024), the results showed that continuous processing mode showed significantly lower latency when using the rate source while maintaining the same throughput. However, when using the Kafka source, continuous processing mode showed worse latency and throughput rates than micro-batch mode.

This surfaced a flaw in initial benchmarking configurations, as Kafka produced much higher throughput than expected, so rate source benchmarks should have been adjusted to match Kafka throughput rates.

Also, continuous processing mode's loss in latency to micro-batching indicated that continuous mode is not running optimally and suggested that high volumes of Kafka throughput may overflow the Spark application's ability to process the data promptly, disrupting latencies. The initial benchmarking results are presented in Table 5.

Table 5
Initial benchmarking results

Benchmark	Throughput (rps)	Mean latency (ms)	50p latency (ms)	75p latency (ms)	99p latency (ms)
Rate (10 rps), 3 workers, micro-batch	10	572.79	613	820	1 030
Rate (10 rps), 3 workers, continuous	10	2.0	0	1	2
Rate (100 rps), 16 workers, micro-batch	99	528.33	528	778	1 018
Rate (100 rps), 16 workers, continuous	100	3.12	0	0	2
Kafka, 3 workers, micro-batch	119 896	388.78	138	393	2 165
Kafka, 3 workers, continuous	32 776	57 219.35	58 824	82 833	105 997

continuous					
Kafka, 16 workers, micro-batch	116 258	197.36	73	104	1496
Kafka, 16 workers, continuous	33 320	55 447.06	55 496	81 386	104 553

Three additional configurations for Kafka source were performed to validate if the number of Kafka topic partitions impacted bad continuous processing performance, the results of which are shown in Table 6.

None of those configurations performed significantly better than the initial ones, proving that despite working with a possibly non-optimal number of Kafka topic partitions, the reason for bad continuous processing performance may be related to other issues, such as throughput overflow.

Table 6
Additional Kafka source benchmarking results

Benchmark	Throughput (rps)	Mean latency (ms)	50p latency (ms)	75p latency (ms)	99p latency (ms)
Kafka (3 partitions), 3 workers, continuous	30 757	56 338.7	57 095	82 433	106 619
Kafka (16 partitions), 16 workers, continuous	29 641	57 040.19	57 881	83 241	107 732
Kafka (48 partitions), 16 workers, continuous	29 861	46 237.82	46 959	67 974	87 808

When analyzing the sustainable throughput benchmarking results of the micro-batching mode shown in Figure 1, benchmarks of 3-worker Spark are similar to 16-worker Spark, which infers that Structured Streaming micro-batch mode works well as long as number of Spark's partitions matches the number of Kafka's partitions. Also, there is a noticeable correlation between micro-batch mode latency, which gets slightly better with an increase in throughput but remains in the range from 510 to 570 ms.

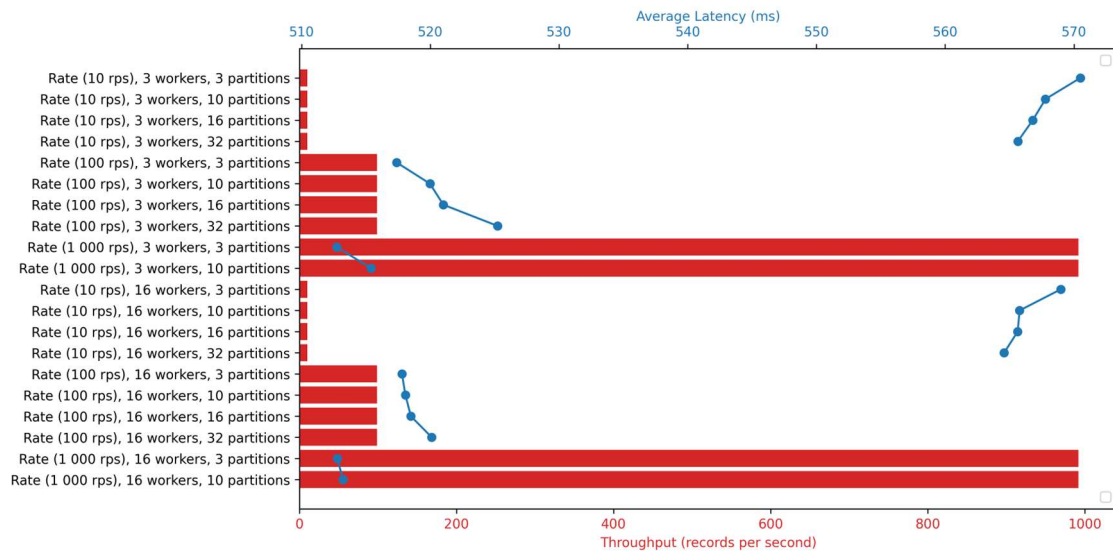


Figure 1: Throughput and latency results of the micro-batching mode using a rate source and a Kafka sink, with the Kafka partition count matching the Spark partition count.

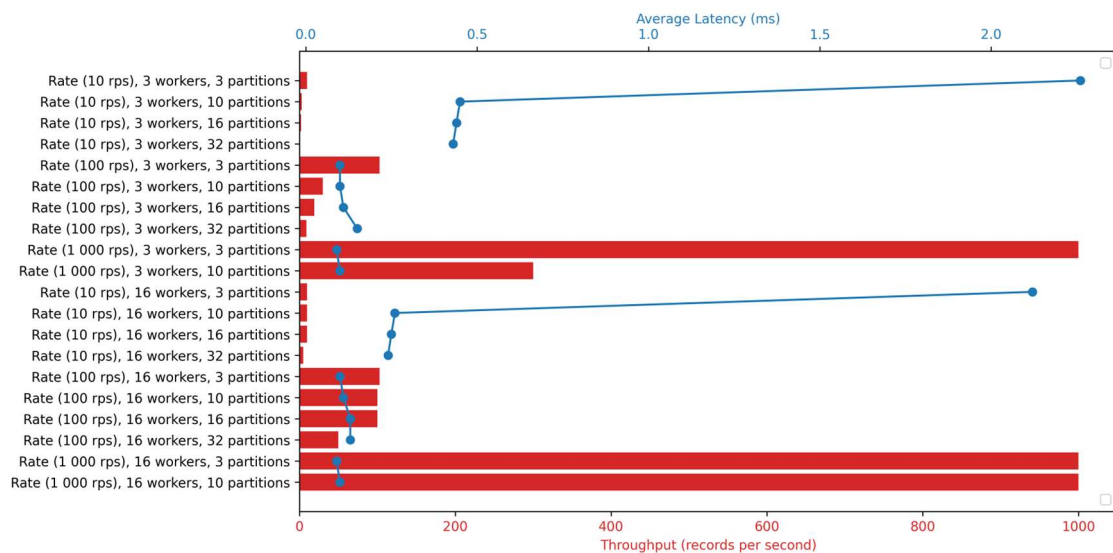


Figure 2: Throughput and latency results of the continuous processing mode using rate source and Kafka sink, with Kafka partition count matching Spark partition count.

On the other hand, results presented in Figure 2 clearly show that the continuous processing mode is much more sensitive to Spark's configuration. Despite generally having sub-millisecond latency (except the configuration with 10 rps and 3 partitions, where the average latency reaches over 2 ms), its throughput suffers greatly from configurations where the number of partitions exceeds the number of workers.

As a result, it is important to analyze existing worker count and partition count configurations when transitioning from micro-batching mode to continuous processing, as

continuous processing launches long-running tasks, which continuously work with a single partition. Therefore, not having enough tasks will leave some partitions unprocessed, which may lead to substantial data loss that may be mistaken as a throughput drop.

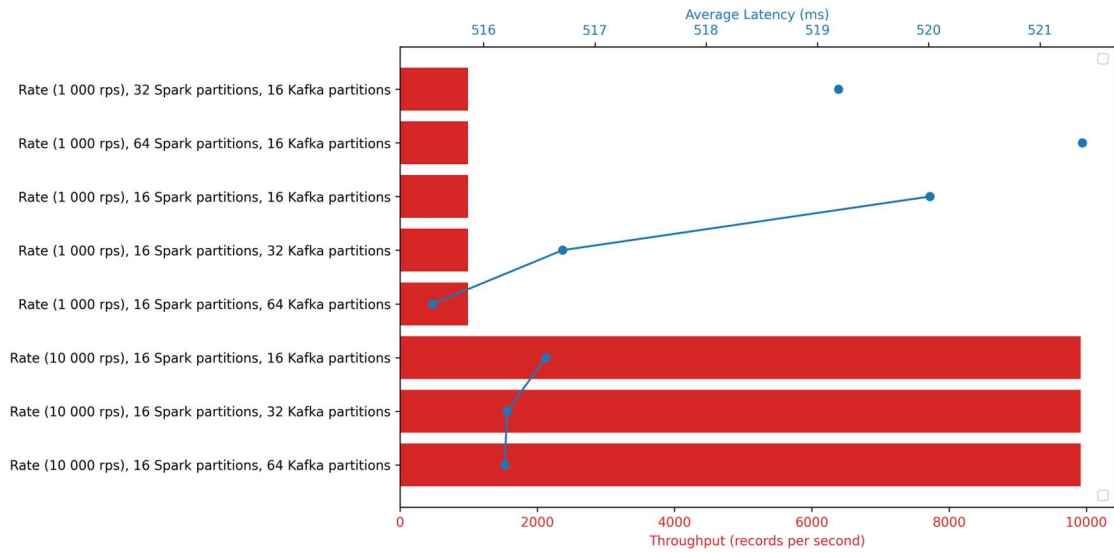


Figure 3: Throughput and latency results of the micro-batching mode with varying Kafka and Spark partition counts.

After observing the benchmark results depicted in Figures 1 and 2, it was decided to expand the research by decoupling the Spark partition count from the Kafka partition count to see if further improvements could be made. Results of micro-batching mode benchmarking are shown in Figure 3, and continuous mode in Figure 4.

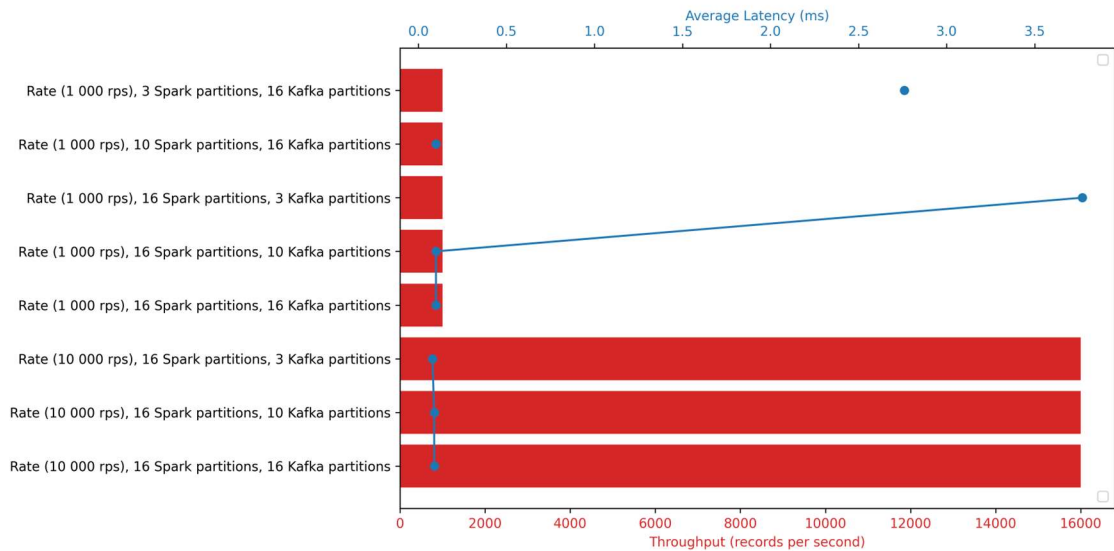


Figure 4: Throughput and latency results of the continuous processing mode with varying Kafka partition count matching Spark partition count.

Throughput and latencies did not significantly change, and for further benchmarks, 16 Spark partitions and 16 Kafka partitions were decided to be the most suitable configurations for both micro-batching and continuous modes. However, during the benchmark for continuous mode at a rate source set to 10 000 rps, the throughput resulted in 16 000 rps, and after investigating the issue, it was concluded that the rate source might be unreliable starting at 10 000 rps for continuous processing mode. Additional tests in the 1 001 – 9 999 rps range were not performed, so the reliability of the rate source at that throughput level is questionable.

The instability of the rate source for continuous processing was further confirmed by conducting a benchmark for 100 000 rps, where micro-batching mode showed 100 000 rps and continuous mode showed 1 063 758 (more than one million) rps, which exceeds the expected throughput by ten times. The investigation established that the issue does not arise from at-least-once delivery guarantees, as no duplicate record values were present in the Kafka sink, meaning each record was unique. Average latencies concluded 537 milliseconds at 100 000 records/s for micro-batch and 41 297 milliseconds at 1 063 758 records/s for continuous mode.

After doing comprehensive research on the rate source and tuning optimal parameters for continuous processing, the last benchmark was attempted to once again measure continuous mode latencies in a simulation close to a real-world use case. Because of bad continuous processing mode performance on high workload levels, the file source in Kafka Connect was replaced by a custom script that implements rate-limiting data read from a corpus on a row count basis. The benchmark was executed on 1 000, 10 000, and 100 000 rps, with partition count in both Kafka and Spark set to 16, with 16 workers. Benchmarking results are provided in Table 7.

Table 7

Kafka source and sink benchmark for continuous processing

Benchmark	Throughput (rps)	Mean latency (ms)	50p latency (ms)	75p latency (ms)	99p latency (ms)
Kafka, 1 000 rps	1 025	555.36	528	531	3 178
Kafka, 10 000 rps	10 238	260.24	104	136	3 351
Kafka, 100 000 rps	24 877	13 156.56	12 315	18 289	24 446

The results have shown that given adjusted parameters for continuous processing mode and rate-limited throughput to 100 000 rps, the throughput got worse by around 20% while improving the mean latency by around 310%. Rate-limiting source to 10 000 rps produced a mean latency of 260 ms, the lowest latency achieved for a Kafka source in continuous mode in the scope of this research.

5. Conclusion

Spark Structured Streaming's Continuous Processing mode prioritizes low-latency data processing, offering users a strategic choice between optimizing throughput and minimizing latency. The platform's use of a declarative API simplifies the construction of robust data pipelines and allows for relatively straightforward toggling between micro-batch and continuous modes. This flexibility is advantageous as it requires minimal code changes,

making it accessible for users to adjust their processing strategies based on evolving data requirements or operational objectives.

The methodology that was proposed in the paper involved creating a simple pipeline supported by micro-batching and continuous processing modes in Apache Spark, using two data sources – Rate source and Kafka topic containing text data from Ukrainian Wikipedia – and varying sources throughput, number of Kafka partitions, and number of Spark workers in order to measure Spark’s throughput and latency.

The results have shown that transitioning between these modes is not merely a matter of code adaptation; it often necessitates tailored adjustments to the Spark configuration. This is due to the inherent architectural distinctions between how micro-batch and continuous processing modes manage data flows and system resources. Each mode is optimized for different aspects of streaming analytics, with micro-batch providing robustness and fault tolerance and continuous processing focusing on reducing processing time to the minimum. The lowest mean latencies achieved for continuous processing mode was 2 ms using rate source (in contrast to 528 ms in micro-batch mode) and 260 ms using Kafka source (in contrast to 197 ms in micro-batch mode).

Furthermore, continuous processing mode must be carefully managed to balance the inherent trade-offs between latency and throughput. In scenarios where the data inflow exceeds the system's processing capacity, there is a significant risk of increased latency, contradicting the primary goal of this mode. Therefore, continuous mode is not universally superior; its effectiveness is contingent upon the specific characteristics and demands of the workload. Conducting thorough performance evaluations, including stress testing under peak data loads, is essential to ensure that the system remains performant and that latency stays within acceptable bounds.

To fully leverage the potential of Continuous Processing mode in Spark Structured Streaming, developers and system architects need to fine-tune configurations and regularly monitor system performance. This proactive approach ensures that the streaming process remains efficient and aligns with their applications' latency and throughput requirements. Ultimately, the choice between micro-batch and continuous processing modes should be informed by a comprehensive understanding of the trade-offs involved and a strategic assessment of the application’s operational priorities.

6. Limitations and Further Research

Further research could be conducted to reach sub-millisecond latencies in the Kafka source instead of the rate source. The relation between Kafka topic message format (Avro, JSON, Protobuf, etc.) and Continuous Processing latency may be investigated. Also, as the benchmarks were conducted on one machine to eliminate possible latency overheads from the networking side, benchmarks could be tested on standalone clusters. Benchmarking pipelines with user-defined functions (UDFs) and MLLib also could be an interesting topic for extending research.

References

- [1] H. Isah, T. Abughofa, S. Mahfuz, D. Ajerla, F. Zulkernine, S. Khan, A Survey of Distributed Data Stream Processing Frameworks, in: *IEEE Access*, vol. 7, pp. 154300-154316, 2019, doi: 10.1109/ACCESS.2019.2946884.
- [2] E. Mehmood, T. Anees, Challenges and Solutions for Processing Real-Time Big Data Stream: A Systematic Literature Review, in: *IEEE Access*, vol. 8, pp. 119123-119143, 2020, doi: 10.1109/ACCESS.2020.3005268.
- [3] J. Torres, M. Armbrust, T. Das, S. Zhu, Introducing Low-latency Continuous Processing Mode in Structured Streaming in Apache Spark 2.3, 2018. URL: <https://www.databricks.com/blog/2018/03/20/low-latency-continuous-processing-mode-in-structured-streaming-in-apache-spark-2-3-0.html>.
- [4] M. Armbrust, T. Das, J. Torres, B. Yavuz, S. Zhu, R. Xin, A. Ghodsi, I. Stoica, and M. Zaharia, Structured Streaming: A Declarative API for Real-Time Applications in Apache Spark, in: *Proceedings of the 2018 International Conference on Management of Data (SIGMOD '18)*. Association for Computing Machinery, New York, NY, USA, 2018, pp. 601–613. URL: <https://doi.org/10.1145/3183713.3190664>.
- [5] Y. Drohobytskiy, V. Brevus, Y. Skorenkyy, Spark Structured Streaming: Customizing Kafka Stream Processing, in: *2020 IEEE Third International Conference on Data Stream Mining & Processing (DSMP)*, Lviv, Ukraine, 2020, pp. 296-299, doi: 10.1109/DSMP47368.2020.9204304.
- [6] H. Mcheick, Y. D. F. Petrillo, S. Ben-Ali, Quality Model for Evaluating and Choosing a Stream Processing Framework Architecture, in: *2019 IEEE/ACS 16th International Conference on Computer Systems and Applications (AICCSA)*, Abu Dhabi, United Arab Emirates, 2019, pp. 1-7, doi: 10.1109/AICCSA47632.2019.9035283.
- [7] J. Karimov, T. Rabl, A. Katsifodimos, R. Samarev, H. Heiskanen, V. Markl, Benchmarking Distributed Stream Data Processing Systems, in: *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, Paris, France, 2018, pp. 1507-1518, doi: 10.1109/ICDE.2018.00169.
- [8] G. Van Dongen, D. Van den Poel, Evaluation of Stream Processing Frameworks, in: *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 8, pp. 1845-1858, 1 Aug. 2020, doi: 10.1109/TPDS.2020.2978480.
- [9] G. Van Dongen, D. Van den Poel, A Performance Analysis of Fault Recovery in Stream Processing Frameworks, in: *IEEE Access*, vol. 9, pp. 93745-93763, 2021, doi: 10.1109/ACCESS.2021.3093208.
- [10] G. Van Dongen, D. Van den Poel, Influencing Factors in the Scalability of Distributed Stream Processing Jobs, in: *IEEE Access*, vol. 9, pp. 109413-109431, 2021, doi: 10.1109/ACCESS.2021.3102645.