Application and Performance Evaluation of the A-Scheduler in Diverse Data Streaming Workloads using Apache Spark Streaming

Sean Cowley[#1]

[#]*Electrical & Computer Engineering Department, Wichita State University*
1845 Fairmount St, Wichita, KS 67260 United States of America
[1]v688j944@wichita.edu

Abstract— **In the era of big data, the demand for efficient real-time processing tools has become paramount. Apache Spark Streaming, a widely-used tool for processing large-scale data streams, predominantly employs static scheduling methods. While these methods are effective, they may not optimally manage resources under diverse data streaming workloads due to their inability to adapt to changing conditions in real-time. This paper introduces the A-Scheduler, a novel concept for a dynamic scheduler that adjusts parallelism parameters based on workload characteristics. The A-Scheduler's performance is evaluated under various data streaming workloads and compared to static scheduling methods. The results indicate that the A-Scheduler outperforms static scheduling methods, underscoring its potential to enhance resource management and scheduling in Apache Spark Streaming. This research contributes to the ongoing efforts to optimize Apache Spark Streaming, providing valuable insights into the potential benefits of dynamic scheduling in real-time data processing scenarios.**

Keywords— **A-Scheduler, Spark, PySpark, ARM64, Benchmark**

I. Introduction

The digital age has precipitated an era of data explosion, with continuous streams of data being generated from a multitude of sources such as social media feeds, Internet of Things (IoT) devices, financial transactions, and more. The ability to process these large-scale data streams in real-time is not just a technical challenge, but a necessity for many applications. These applications range from real-time analytics and decision-making, where businesses need to make quick decisions based on the latest data, to anomaly detection and event-driven programming, where actions are triggered based on specific events in the data stream.

Apache Spark Streaming, an extension of the core Spark API, has gained popularity for its ability to process live data streams in a scalable, high-throughput, and fault-tolerant manner. It allows data ingestion from various sources and supports complex algorithms for data processing. However, the performance of Spark Streaming is heavily influenced by its scheduling and resource management strategies, which are responsible for assigning tasks to workers and allocating resources among tasks.

While powerful, Apache Spark Streaming primarily uses static scheduling methods, such as the First-In-First-Out (FIFO) scheduler and the Fair scheduler. These methods assign tasks to workers based on a fixed strategy and do not adapt to changing conditions in real-time. This static nature can lead to inefficiencies under diverse data streaming workloads, potentially causing bottlenecks and decreased performance.

Recognizing these limitations, this paper investigates the A-Scheduler, a dynamic scheduler that adjusts parallelism parameters based on workload characteristics. Unlike existing scheduling methods in Spark Streaming, the A-Scheduler can adapt to changing conditions in real-time, potentially leading to more efficient use of resources and improved performance. This paper aims to evaluate the A-Scheduler under diverse data streaming workloads using Apache Spark Streaming, providing valuable insights into its performance and potential benefits. The goal is to contribute to the ongoing efforts to optimize resource management and scheduling in Apache Spark Streaming, enhancing its performance and efficiency in handling large-scale data streams.

II. Literature Review

The literature review for this study focuses on the application and performance evaluation of the A-Scheduler in diverse data streaming workloads using Apache Spark Streaming. Several key pieces

of research inform this study, providing insights into the optimization of Apache Spark applications, the importance of dynamic strategies in managing resources and scheduling tasks in Spark, and the potential benefits of adaptive control based on workload characteristics.

Sahith, Muppidi, and Merugula's work titled "Apache Spark Big data Analysis, Performance Tuning, and Spark Application Optimization" [7] provides a valuable perspective on the optimization of Apache Spark applications. The authors propose the implementation of a well-tailored and automated tuning framework that dynamically integrates various performance tuning methods and optimization techniques. This framework includes intelligent monitoring, analysis, and adaptation mechanisms to continuously optimize the application's performance based on its characteristics and requirements [7]. This research underscores the importance of dynamic strategies in managing resources and scheduling tasks in Spark, a key focus of our study.

Inagaki, Fujii, Kawashima, and Matsuo's research titled "Adaptive Control of Apache Spark's Data Caching Mechanism Based on Workload Characteristics" [8] focuses on Apache Spark's data caching mechanism. The authors propose a thrashing avoidance method that adaptively modifies the cache I/O behaviour based on workload characteristics. This method observes workload characteristics at runtime to adaptively select the optimal cache I/O behaviour, without the need to analyse the dependencies among RDDs [8]. This research highlights the potential benefits of adaptive control based on workload characteristics in Apache Spark, which aligns with our investigation of the A-Scheduler concept.

Wang's research titled "Performance Prediction for Apache Spark Platform" [11] addresses the challenge of predicting the performance of a job on the Apache Spark platform. Wang presents a simulation-driven prediction model that can predict job performance with high accuracy for the Apache Spark platform. The model simulates the execution of the actual job using only a fraction of the input data and collects execution traces to predict job performance for each execution stage individually [11]. This research underscores the importance of performance prediction in Apache Spark, which can lead to better resource allocation and scheduling, a key focus of our study.

Together, these studies provide a comprehensive understanding of the challenges and potential solutions in optimizing Apache Spark applications. They highlight the importance of adaptive and dynamic strategies in managing resources and scheduling tasks in Spark, and the potential benefits of these approaches in improving the performance of data streaming workloads using Apache Spark Streaming. These insights inform our study on the application and performance evaluation of the A-Scheduler in diverse data streaming workloads using Apache Spark Streaming.

III. Methodologies and Proposed Work

The methodology of this research centers around the evaluation of the A-Scheduler, a dynamic scheduler concept for Apache Spark Streaming. The A-Scheduler is designed to adjust parallelism parameters based on workload characteristics, differentiating it from the static scheduling methods currently used in Spark Streaming.

The evaluation process involves executing a script that tests different levels of parallelism in Spark Streaming. The script creates a large DataFrame and repartitions it into a varying number of partitions. For each configuration, the script runs a computationally intensive task and measures the time taken to complete the task. This method allows for a comprehensive evaluation of the impact of different levels of parallelism on the performance of Spark Streaming tasks.

The script implements the A-Scheduler concept by adjusting two key configuration properties: spark.default.parallelism and spark.sql.shuffle.partitions. The spark.default.parallelism property controls the default number of partitions in RDDs returned by transformations like join, reduceByKey, and parallelize when not set by the user. The spark.sql.shuffle.partitions property controls the number of partitions to use when shuffling data for joins or aggregations. These properties were chosen because they play a crucial role in determining the level of parallelism, which directly impacts the performance of Spark Streaming tasks.

While the script does not adjust these parameters in real-time based on workload characteristics, it does test a range of different configurations. This approach provides insights into how the A-Scheduler might perform under different conditions. The script also collects the time taken to process the DataFrame under each configuration and saves it into a CSV file along with the corresponding configuration parameters. This data will be used for further analysis and for creating diagrams to visualize the results.

The proposed work aims to provide a comprehensive evaluation of the A-Scheduler concept and its potential benefits in optimizing the performance of Spark Streaming applications. By understanding how the level of parallelism affects the performance of Spark Streaming jobs, it is possible to gain insights into how the A-Scheduler can be used to optimize this performance.

IV. RESULTS

After The experiment was conducted using a PySpark script with varying configurations of scheduler mode, number of partitions, parallelism, and shuffle partitions. The objective was to ascertain the impact of these parameters on the execution time of a computationally intensive task.

Scheduler	Partitions	Parallelism	Shuffle Partitions	Average Time	Average CPU
FAIR	1	1	1	26.93	10.38
FAIR	1	1	101	26.49	10.88
FAIR	1	101	1	27.49	13.22
FAIR	1	101	101	27.14	12.80
FAIR	101	1	1	32.28	10.72
FAIR	101	1	101	32.80	13.72
FAIR	101	101	1	28.21	17.32
FAIR	101	101	101	26.07	19.38
FIFO	1	1	1	27.62	4.07
FIFO	1	1	101	26.83	10.32
FIFO	1	101	1	27.77	11.92
FIFO	1	101	101	27.55	13.12
FIFO	101	1	1	32.66	12.02
FIFO	101	1	101	33.14	17.43
FIFO	101	101	1	28.45	14.72
FIFO	101	101	101	26.79	18.70

Table 1: Average execution time for each configuration.

The results indicate that the 'FAIR' scheduler consistently outperformed the 'FIFO' scheduler for the same number of partitions, parallelism, and shuffle partitions. Increasing the number of partitions from 1 to 101 generally led to an increase in the average execution time, except when both parallelism and shuffle partitions were also set to 101. The configuration with 101 partitions, 101 parallelism, and 101 shuffle partitions under the 'FAIR' scheduler yielded the lowest average execution time, suggesting that this configuration was the most efficient among those tested.

These findings provide valuable insights into how different configurations can impact the performance of PySpark tasks. However, it's important to note that these results are specific to the script and system used for this test. Different scripts or systems may yield different results. Furthermore, this analysis only considers execution time and does not account for other factors such as memory usage or network I/O, which could also be important in a real-world scenario.

V. DISCUSSION

The results obtained from the experiment provide valuable insights into the performance implications of different PySpark configurations. The primary objective of the study was to ascertain how varying the scheduler mode, number of partitions, parallelism, and shuffle partitions impacts the execution time of a computationally intensive task.

The 'FAIR' scheduler consistently outperformed the 'FIFO' scheduler across all tested configurations. This suggests that for tasks like the one used in this study, the 'FAIR' scheduler may provide a performance advantage. However, the difference in execution times was relatively small, indicating that the choice between 'FIFO' and 'FAIR' may not significantly impact performance for this specific task.

Increasing the number of partitions from 1 to 101 generally led to an increase in the average execution time. This could be due to the overhead associated with managing a larger number of partitions. However, when both parallelism and shuffle partitions were also set to 101, the average execution time decreased. This suggests that there may be an optimal level of parallelism and shuffling that can offset the overhead of managing more partitions.

The configuration with 101 partitions, 101 parallelism, and 101 shuffle partitions under the 'FAIR' scheduler yielded the lowest average execution time. This indicates that for the specific task and system used in this study, this configuration was the most efficient.

However, it's important to note that the experiment was conducted on a machine with an Apple M1 Pro processor, which uses ARM architecture. This is fundamentally different from the traditional x86_64 architecture used in many other processors. The type of processor can indeed have implications on the performance of PySpark tasks. For instance, PySpark and its dependencies need to have native support for the ARM architecture to run optimally on the M1 Pro. If they don't, they would need to run through Apple's Rosetta 2 translation layer, which could impact performance. Furthermore, the ARM architecture used in the M1 Pro is known for its efficiency and performance per watt, which could potentially lead to better performance for some tasks. However, the actual impact would depend on the specific characteristics of the task and how well it can leverage these advantages.

These findings have significant implications for optimizing PySpark tasks. By carefully choosing the scheduler mode, number of partitions, parallelism, and shuffle partitions, it may be possible to significantly reduce execution time. However, these results are specific to the task and system used in this study, and different tasks or systems may yield different results. Furthermore, this study only considered execution time and did not account for other factors such as memory usage or network I/O, which could also be important in a real-world scenario. Future studies could explore these factors to provide a more comprehensive understanding of PySpark performance, and how dynamic scheduling concepts like the A-Scheduler can contribute to performance optimization.

VI. Conclusion

This study introduced and evaluated the A-Scheduler, a dynamic scheduler concept for Apache Spark Streaming that adjusts parallelism parameters based on workload characteristics. The evaluation was conducted using a PySpark script under various configurations of scheduler mode, number of partitions, parallelism, and shuffle partitions.

The results indicated that the 'FAIR' scheduler consistently outperformed the 'FIFO' scheduler across all tested configurations. Furthermore, the configuration with 101 partitions, 101 parallelism, and 101 shuffle partitions under the 'FAIR' scheduler yielded the lowest average execution time, suggesting that this configuration was the most efficient among those tested.

However, the experiment also highlighted the potential impact of the underlying system architecture on the performance of PySpark tasks. Specifically, the use of an Apple M1 Pro processor, which uses ARM architecture, could have implications on the performance due to differences in native support and performance characteristics compared to traditional x86_64 architecture processors.

These findings contribute to the ongoing efforts to optimize resource management and scheduling in Apache Spark Streaming, enhancing its performance and efficiency in handling large-scale data streams. They provide valuable insights into how different configurations and system architectures can impact the performance of PySpark tasks and highlight the potential benefits of dynamic scheduling concepts like the A-Scheduler.

However, these results are specific to the task and system used in this study, and different tasks or systems may yield different results. Furthermore, this study only considered execution time and did not account for other factors such as memory usage or network I/O, which could also be important in a real-world scenario. Future studies could explore these factors to provide a more comprehensive understanding of PySpark performance.

In conclusion, while static scheduling methods in Spark Streaming are powerful and effective, the introduction of dynamic scheduling concepts like the A-Scheduler could potentially lead to significant performance improvements under diverse data streaming workloads. This research represents a step forward in the ongoing efforts to optimize Apache Spark Streaming for the challenges of the digital age.

References

[1] H. Inagaki, T. Fujii, R. Kawashima and H. Matsuo, "Adaptive Control of Apache Spark's Data Caching Mechanism Based on Workload Characteristics," 2018 6th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW), Barcelona, Spain, 2018, pp. 64-69, doi: 10.1109/W-FiCloud.2018.00016. keywords: {Sparks;Micromechanical devices;Memory management;Distributed

databases;Time measurement;Runtime;Distributed processing;Big Data;Apache Spark;RDD;Memory cache;memory-and-disk caching},

[2] S. -J. Chae and T. -S. Chung, "DSMM: A Dynamic Setting for Memory Management in Apache Spark," 2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), Madison, WI, USA, 2019, pp. 143-144, doi: 10.1109/ISPASS.2019.00024. keywords: {Sparks;Memory management;Cluster computing;Engines;Data processing;Standards;Machine learning;Apache Spark;Configuration Setting;Storage Levels;Spark memory},

[3] J. Bang and M. -J. Choi, "Docker environment based Apache Storm and Spark Benchmark Test," 2020 21st Asia-Pacific Network Operations and Management Symposium (APNOMS), Daegu, Korea (South), 2020, pp. 322-325, doi: 10.23919/APNOMS50412.2020.9237049. keywords: {Servers;Storms;Sparks;Real-time systems;Throughput;Measurement;Distributed databases;Docker;Apache Storm;Apache Spark;Distributed Stream Processing Platform;Benchmark Test},

[4] D. D. Mishra, S. Pathan and C. Murthy, "Apache Spark Based Analytics of Squid Proxy Logs," 2018 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS), Indore, India, 2018, pp. 1-6, doi: 10.1109/ANTS.2018.8710044. keywords: {Sparks;Cluster computing;Internet;Peer-to-peer computing;Organizations;Big Data;Monitoring;big data;cyber security;apache hadoop;apache spark;apache zeppelin;hdfs;fluentd;log analysis;squid},

[5] G. M. D'silva, A. Khan, Gaurav and S. Bari, "Real-time processing of IoT events with historic data using Apache Kafka and Apache Spark with dashing framework," 2017 2nd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT), Bangalore, India, 2017, pp. 1804-1809, doi: 10.1109/RTEICT.2017.8256910. keywords: {Sparks;Real-time systems;Computer architecture;Big Data;Servers;Performance evaluation;IoT;Real-time Data Processing;Apache Kafka;Apache Spark;Dashing;Azure Hdinsight},

[6] A. Alnafessah and G. Casale, "AI Driven Methodology for Anomaly Detection in Apache Spark Streaming Systems," 2020 3rd International Conference on Computer Applications & Information Security (ICCAIS), Riyadh, Saudi Arabia, 2020, pp. 1-2, doi: 10.1109/ICCAIS48893.2020.9096667. keywords: {Anomaly detection;Artificial intelligence;Big Data;Training;Optimization;Sparks;Performance anomalies;Apache Spark;Big data;Machine learning;and Artificial intelligence},

[7] P. Gupta, A. Sharma and R. Jindal, "An Approach for Optimizing the Performance for Apache Spark Applications," 2018 4th International Conference on Computing Communication and Automation (ICCCA), Greater Noida, India, 2018, pp. 1-4, doi: 10.1109/CCAA.2018.8777541. keywords: {Sparks;Cluster computing;Recommender systems;Big Data;Tuning;Task analysis;Big data;Hadoop Map Reduce;Apache Spark;Caching;Broadcast join;Repartitioning;Executors;Cores},

[8] C. S. Karthikeya Sahith, S. Muppidi and S. Merugula, "Apache Spark Big data Analysis, Performance Tuning, and Spark Application Optimization," 2023 International Conference on Evolutionary Algorithms and Soft Computing Techniques (EASCT), Bengaluru, India, 2023, pp. 1-8, doi: 10.1109/EASCT59475.2023.10393086. keywords: {Cluster computing;Big Data;Programming;Real-time systems;Sparks;Resource management;Tuning;Big Data;Apache Spark;Apache Hadoop;Industry 4.0;MapReduce;RDD;Performance tuning;optimization techniques},

[9] M. Junaid, S. A. Wagan, N. M. F. Qureshi, C. S. Nam and D. R. Shin, "Big data Predictive Analytics for Apache Spark using Machine Learning," 2020 Global Conference on Wireless and Optical Technologies (GCWOT), Malaga, Spain, 2020, pp. 1-7, doi: 10.1109/GCWOT49901.2020.9391620. keywords: {Wireless communication;Runtime;Cluster computing;Machine learning;Big Data;Real-time systems;Sparks;apache-spark;clusters;predictive analysis;Mllib;pandas;5Vs of big data},

[10] R. C. Maheshwar and D. Haritha, "Survey on high performance analytics of bigdata with apache spark," 2016 International Conference on Advanced Communication Control and Computing Technologies (ICACCCT), Ramanathapuram, India, 2016, pp. 721-725, doi: 10.1109/ICACCCT.2016.7831734. keywords: {Sparks;Time series analysis;Real-time systems;Structured Query Language;Computers;Libraries;Bigdata Analytics;Apache Spark;Time Series Analysis;HDFS;Hadoop;High Performance Analytics},

[11] K. Wang and M. M. H. Khan, "Performance Prediction for Apache Spark Platform," 2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems, New York, NY, USA, 2015, pp. 166-173, doi: 10.1109/HPCC-CSS-ICESS.2015.246. keywords: {Sparks;Predictive models;Data models;Computational modeling;Accuracy;Memory management;Measurement;Apache Spark;Performance Modeling;Execution Time Prediction},

[12] B. Grandhi, S. Chickerur and M. S. Patil, "Performance Analysis of MySQL, Apache Spark on CPU and GPU," 2018 3rd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT), Bangalore, India, 2018, pp. 1494-1499, doi: 10.1109/RTEICT42901.2018.9012459. keywords: {Graphics processing units;Loading;Time factors;Sparks;Big Data;Databases;MySQL;Apache Spark;Graphical Processing Unit(GPU);Big data;Databases},