Evaluating Lazy and Non-Lazy Execution Strategies in Spark for Big Data Processing Optimization

Thi-Thu-Trang Do[1,2]
[1]Faculty of Information Technology
Hung Yen University of Technology and Education, Hungyen
Vietnam
trangdtt@utehy.edu.vn

Thi-Thuy-Linh Tong
Faculty of Information Technology
Hung Yen University of Technology and Education, Hungyen
Vietnam
thuylinh1432012@gmail.com

Viet-Anh Nguyen
Faculty of Information Technology
Hung Yen University of Technology and Education, Hungyen
Vietnam
vietanhtvym@gmail.com

Manh-Hung Ngo
Vinacomin Mining Chemical Industry Holding Corporation Limited
Hanoi, Vietnam
hungnm@micco.com.vn

Van-Quyet Nguyen*
Faculty of Information Technology
Hung Yen University of Technology and Education, Hungyen
Vietnam
quyetict@utehy.edu.vn

Quyet-Thang Huynh
[2]School of Information and Communication Technology, Hanoi University of Science and Technology
Hanoi, Vietnam
thanghq@soict.hust.edu.vn

ABSTRACT

This study investigates the impact of lazy and non-lazy execution strategies on the performance of big data processing in Apache Spark. We evaluate various Spark operations, comparing their performance under execution strategies using real-world and synthetic datasets. The evaluation emphasizes the importance of matching execution strategies to workload characteristics, data volumes, and query complexity to optimize execution time and resource utilization. Through ten scenarios, the experimental results show that lazy execution, while reducing redundant computations, can be inefficient in frequent iterative or memory-bound tasks. In contrast, non-lazy execution performs better in immediate and incremental computations but incurs higher overhead with overlapping transformations or shared intermediate results. These findings underline the need for a hybrid strategy tailored to workloads, offering actionable guidance for optimizing Spark applications.

KEYWORDS

Lazy Evaluation, Non-Lazy Evaluation, Execution Optimization, Data Processing Strategies, Big Data Processing.

1. INTRODUCTION

Apache Spark is a leading platform for big data analytics, valued for its efficient in-memory processing. A key factor influencing its performance is the evaluation mechanism, specifically the choice between lazy and non-lazy evaluation. Lazy evaluation defers transformations until an action is triggered, enabling global optimization and reducing redundant computations but complicating debugging and predictability. Conversely, non-lazy evaluation processes transform immediately, offering simpler execution but missing optimization opportunities, potentially leading to inefficient resource use. Choosing the appropriate evaluation strategy is critical for optimizing Spark's performance. However, comparing these strategies is challenging due to the complexity of Spark's Directed Acyclic Graph (DAG) optimization and variations in workload characteristics. These challenges underscore the need for a detailed empirical analysis to understand their impact under different conditions.

Previous research has predominantly focused on the benefits of lazy evaluation, emphasizing its role in optimizing execution plans through deferred computation, as shown in studies by Zaharia et al. [1] and Armbrust et al. [2]. These works illustrate how lazy evaluation minimizes redundant processing via DAG construction before execution. However, detailed analysis of non-lazy (eager) evaluation is limited, with most literature highlighting its advantage in immediate feedback for real-time analytics and interactive queries [3]. While practical guides, such as those by Karau and Warren [4], briefly mention non-lazy contexts, there is a lack of systematic empirical comparisons. Our study indicates a need for a detailed assessment of the trade-offs between lazy and non-lazy evaluation across diverse workloads.

The contributions of our paper are as follows:

- Presenting a comprehensive performance analysis of lazy and non-lazy execution strategies in Spark, addressing gaps in the literature with empirical evidence rather than theoretical discussions.
- Evaluating performance through ten experimental scenarios using the real-world Stack Overflow dataset and five system logs datasets. These scenarios cover various transformations and actions, such as filtering, grouping, joining, and sequential operations.
- Providing practical recommendations for selecting the most appropriate evaluation strategy, considering workload characteristics, data volumes, and query complexity to improve the efficiency and performance of Spark applications.

The rest of this paper is structured as follows: Section 2 compares execution strategies in Spark, emphasizing the

mechanisms and key factors of lazy and non-lazy execution. Section 3 describes the experimental setup, datasets, results, and discussion. Section 4 concludes with key findings, recommendations for optimizing Spark applications, and directions for future research.

2. COMPARING EXECUTION STRATEGIES IN SPARK

2.1 Lazy and Non-Lazy Evaluation

In Spark, the evaluation mechanism defines when and how data computations occur, using two strategies: lazy and non-lazy evaluation, each with unique characteristics and applications.

Lazy evaluation defers computation until an action like collect() or count() is triggered. Transformations such as map() or filter() are not executed immediately but added to a DAG representing the logical workflow. This delay enables Spark to optimize the DAG by combining transformations, minimizing data movement, and avoiding unnecessary computations, leading to faster execution and resource efficiency. However, debugging becomes challenging since errors are only revealed during action execution, complicating troubleshooting in complex workflows [5].

Non-lazy evaluation processes transform immediately as defined, offering a clear and predictable execution flow. This simplifies debugging by identifying errors at each step, reduces latency, and suits real-time analytics. Additionally, caching intermediate results enhances efficiency for repetitive tasks. However, the absence of global optimization can cause redundant computations and less efficient resource use in complex workflows [6]. Figure 1 compares log data processing under Lazy and Non-Lazy evaluation, showing differences in transformation and execution approach.

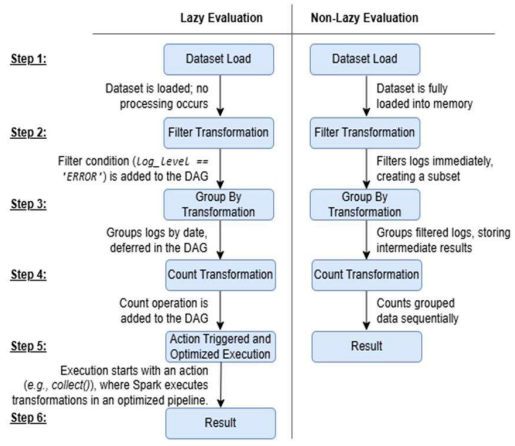


Figure 1: Execution flow comparison of Lazy and Non-Lazy evaluation.

2.2 Choosing between Lazy and Non-Lazy Evaluation: Key Factors

Key factors affect the performance and efficiency of lazy and non-lazy execution strategies selection as shown in Table 1. The choice should consider workload complexity, optimization needs, debugging ease, and resource constraints to achieve optimal performance.

Table 1: Key Factors Affecting Performance and Efficiency between Lazy and Non-Lazy Evaluation

Factors	Lazy Evaluation	Non-Lazy Evaluation
Workload Complexity	Optimizes complex workflows but slower debugging.	Immediate execution but lacks global optimization.
Data Reuse	Recomputes transformations, increasing overhead.	Caches results, efficient for repetitive tasks.
Action Frequency	Efficient for fewer actions, reducing execution overhead.	Handles frequent actions but can increase redundancy.
Data Volume	Processes subsets efficiently, saving resources.	Handles full datasets better, avoiding recomputation.

3. EXPERIMENTS AND RESULTS

3.1 Environment Settings

Our experiments were deployed on a cluster with a master node and two slaves. Each node has 8GB of RAM and 8 CPU cores. This setup provided consistent conditions to compare the performance of lazy and non-lazy evaluation strategies under diverse workloads.

3.2 Datasets

Two datasets were used to evaluate Spark execution strategies. The first one is the Stack Overflow dataset, containing 2.78GB of real-world data with over 13 million records on posts, users, comments, votes, and tags. It provides a comprehensive context for testing complex queries involving joins, aggregations, and filtering and is detailed in Table 2.

Table 2: Summary of Stack Overflow Dataset

Table Name	Description	Records
comments	Contains comments on posts, including creation date, score, and user details.	2,906,704
posts	Stores main content of posts (questions and answers) with associated metadata.	2,623,637
tags	Lists tags used in posts, tracking usage and linked wiki entries.	2,213,139

Table Name	Description	Records
users	Holds user information, such as name, reputation, and activity	2,537,201
votes	Records votes on posts, including type and user data.	3,004,052

The second dataset is a synthetic system logs dataset created to evaluate execution strategies across varying data sizes. Five subsets, ranging from 2GB to 10GB, were generated to test common log analysis tasks, including filtering, grouping, and saving, as detailed in Table 3.

Table 3: Summary of System Logs Datasets

Dataset	Records	Size
D1	36,800,000	2GB
D2	73,600,000	4GB
D3	110,400,000	6GB
D4	147,200,000	8GB
D5	184,000,000	10GB

3.3 Experimental Scenarios

We conducted ten experimental scenarios to evaluate the performance of the two execution strategies. Six scenarios, detailed in Table 4, focused on workloads using the Stack Overflow dataset, while four scenarios, presented in Table 5, examined workloads on log analysis with the system logs dataset.

Table 4: Experimental Scenarios for Stack Overflow Dataset

Experiment	Description
Exp-1	Count posts and comments per user, along with the average comment score.
Exp-2	Retrieve posts with over 10 comments and an average comment score above 5.
Exp-3	Calculate the number of comments per post.
Exp-4	Identify users with upvotes and analyze their commenting activity.
Exp-5	Find users with over 100 posts, analyze post count, and retrieve profile details.
Exp-6	Analyze popular tags, calculate average post scores, and retrieve user details for these tags.

Table 5: Experimental Scenarios for System Logs Datasets

Experiment	Description
Exp-7	Extract dates, filter ERROR logs, count daily errors, and display sorted results.
Exp-8	Group logs by date and level, calculate counts, and display statistics.
Exp-9	Filter ERROR logs, count by date and level, identify high-error days, and save to HDFS.
Exp-10	Extract, filter, count, compute totals, and save logs to HDFS.

3.4 Experimental Results

Figure 2 illustrated the performance comparison between lazy and non-lazy evaluations on the Stack Overflow dataset. We

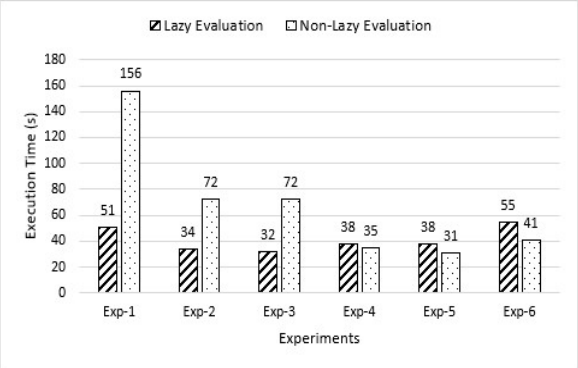
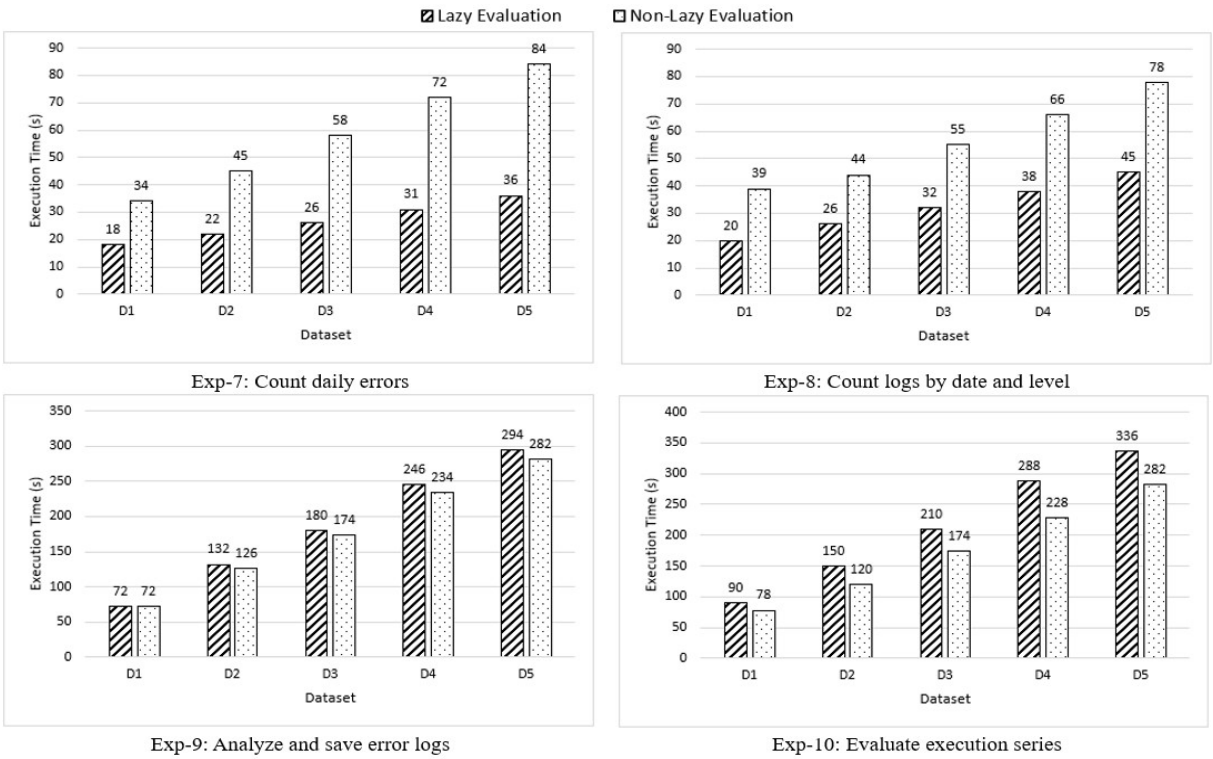


Figure 2: Performance Comparison on Stack Overflow Dataset

observe that lazy evaluation outperforms non-lazy evaluation in a complex workflow (e.g., Exp-1) and simple workloads (e.g., Exp-2 and Exp-3). By leveraging DAG optimization to minimize redundant computations, lazy evaluation achieves performance up to three times faster than non-lazy evaluation. In contrast, Exp-4, Exp-5, and Exp-6, with straightforward workflows or frequent reuse of intermediate results, demonstrate improved performance under non-lazy evaluation. The reason is that these workloads used immediate executions and cached results. Therefore, we can reduce latency and eliminate the overhead of deferred computation.

Figure 3 demonstrated the performance comparison between lazy and non-lazy evaluation on the system logs datasets. In Exp-7 and Exp-8, lazy evaluation achieves the average execution times faster than non-lazy evaluation up to 2.2 times. In these experiments, lazy evaluation optimizes data shuffling and eliminates redundant computations. In contrast, non-lazy evaluation exhibits superior efficiency in Exp-9 and Exp-10, where its immediate execution reduces deferred computation overhead. Notably, in Exp-10, as the data volume grows, non-lazy evaluation demonstrates enhanced scalability, further improving its effectiveness for sequential workflows with large datasets.

3.5. Discussion



Exp-7: Count daily errors

Exp-8: Count logs by date and level

Exp-9: Analyze and save error logs

Exp-10: Evaluate execution series

Figure 3: Performance Comparison between Lazy and Non-Lazy Evaluation on the System Logs Dataset

The experimental results from Figures 2 and 3 highlight the complementary strengths of lazy and non-lazy evaluation strategies, providing practical guidance for optimizing Spark-based applications. Lazy evaluation proves highly effective for complex workflows by leveraging DAG optimization to minimize redundant computations and data shuffling. This is evident in scenarios such as Exp-1 and Exp-7, where lazy evaluation achieves execution times up to three times faster than non-lazy evaluation. Even in simple workloads such as Exp-2 and Exp-8, lazy evaluation maintains an edge due to its optimization capabilities. Conversely, non-lazy evaluation excels in workloads requiring immediate feedback or frequent reuse of intermediate results. Scenarios such as Exp-4 and Exp-9 benefit from their step-by-step execution, avoiding the overhead of deferred computation. Additionally, non-lazy evaluation demonstrates better scalability for sequential workflows with large datasets, as shown in Exp-10. These findings suggest that lazy evaluation is well-suited for resource-intensive, complex workflows, while non-lazy evaluation is more effective for latency-sensitive workloads and repetitive operations. Given these findings, a hybrid execution strategy emerges as a compelling solution to optimize performance across diverse workloads. For instance, lazy evaluation can be employed for preprocessing stages involving heavy transformations, while non-lazy execution can be activated for subsequent stages requiring immediate results or iterative feedback. This combination can balance the trade-offs between resource utilization and latency, ensuring both efficiency and responsiveness.

4. CONCLUSIONS

This paper evaluated lazy and non-lazy execution strategies in Spark, highlighting their strengths across different scenarios. The lazy evaluation demonstrated superior performance for workloads with complex workflows or a few actions by leveraging global optimization and minimizing redundant computations. In contrast, non-lazy evaluation excelled in workloads simple workflows with multiple actions on large datasets by leveraging cached data in memory. These findings emphasize the importance of selecting an evaluation strategy based on workload characteristics such as data volume and query complexity. This study uniquely bridges the gap in the literature by offering a systematic comparison of lazy and non-lazy strategies, complemented by actionable hybrid solutions.

In the future work, we aim to explore the impact of lazy and non-lazy evaluation strategies on resource utilization, particularly memory and CPU efficiency. We also plan to evaluate their performance in distributed environments with varying cluster configurations. These efforts aim to assist in optimizing Spark performance across diverse real-world deployment scenarios, contributing to the broader field of big data processing and distributed computing.

ACKNOWLEDGMENTS

This research was supported by Vietnam National Coal and Mineral Industries Group under grant number KC.02.Đ07-23/21-25. This research was supported by Hung Yen University of Technology and Education under the grant number UTEHY.L.2023.02.

REFERENCES

[1] Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., & Stoica, I. (2016). Apache Spark: A Unified Engine for Big Data Processing. *Communications of the ACM*, 59(11), 56–65. DOI: http://dx.doi.org/10.1145/2934664

[2] Armbrust, M., Xin, R. S., Lian, C., Huai, Y., Liu, D., Bradley, J., & Zaharia, M. (2015). Spark SQL: Relational Data Processing in Spark. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data (SIGMOD '15)*, 1383–1394. DOI: http://dx.doi.org/10.1145/2723372.2742797

[3] Laskowski, J. (2023). Apache Spark can be eagerly evaluated too: Commands on waitingforcode.com. Retrieved from https://www.waitingforcode.com/apache-spark-sql/apache-spark-eagerly-evaluated-commands/read.

[4] Karau, H., & Warren, R. (2017). High Performance Spark: Best Practices for Scaling and Optimizing Apache Spark. O'Reilly Media, Inc. Retrieved from https://www.oreilly.com/library/view/high-performance-spark/9781491943205/

[5] MSSQLTips. (2023). Apache Spark's DAG & Lazy Evaluation: Optimizing Performance. Retrieved from https://www.mssqltips.com/sqlservertip/7897/apache-sparks-dag-lazy-evaluation-optimizing-performance/

[6] Scaler Topics. (2023). Lazy Evaluation in Spark. Retrieved from https://www.scaler.com/topics/lazy-evaluation-in-spark/