ISBN: 978-981-94-1156-6
2024 the 14th International Workshop on Computer Science and Engineering (WCSE 2024)
doi: 10.18178/wcse.2024.06.015

On the Use of Data Parallelism Technologies for Implementing Statistical Analysis Functions

Amirkia Rafiei Oskooei [1, 2 +]

[1] R&D Center, Intellica Business Intelligence Consultancy, Istanbul, Turkey
[2] Computer Engineering Dept., Yildiz Technical University, Istanbul, Turkey

Abstract. This study presents a comparative analysis of data parallelism technologies for implementing statistical analysis functions using the Apache Spark big data processing framework. As data volume and complexity continues to grow exponentially, selecting the right parallel processing framework is crucial for efficient big data analysis. Through a comprehensive methodology, we evaluate the performance and suitability of Spark's data parallelism capabilities for implementing descriptive, exploratory, and inferential statistical functions. By comparing Apache Spark with Hadoop MapReduce, the study highlights Spark's superior performance, especially in handling complex and iterative analytical tasks. The findings show significant performance gains with Spark, positioning it as the preferred framework for a variety of statistical analysis needs in the big data era. The findings of this research offer valuable insights for researchers and practitioners looking to optimize their data analysis workflows and leverage the full potential of big data technologies.

Keywords: Big Data Analytics, Statistical Functions, Apache Spark, Data Parallelism, MapReduce, Parallel Processing

1. Introduction

The exponential growth of data in recent years, fueled by technological advancements, has initiated the era of Big Data. This vast and diverse data presents enormous opportunities across various industries, from healthcare and finance to marketing and social media [1]. However, extracting meaningful insights from Big Data requires sophisticated analysis techniques that can handle its inherent challenges. Statistical analysis plays a crucial role in unlocking the potential of Big Data. Descriptive statistics provide summaries and key characteristics of the data, while exploratory analysis helps uncover patterns and relationships. Inferential statistics allow us to draw conclusions about larger populations based on samples, leading to actionable insights [2]. However, traditional statistical methods struggle with the sheer size and complexity of Big Data, necessitating advanced solutions. Fortunately, technological advancements have also brought innovative data processing models like MapReduce and Spark. These distributed computing frameworks enable parallel processing across multiple nodes, significantly improving the performance and scalability of Big Data analysis. Despite its power, MapReduce suffers limitations in handling complex analytical tasks and iterative algorithms. This has paved the way for Spark, a more versatile framework offering in-memory computing and a rich set of libraries for efficient data manipulation and analysis [3] [4].

To establish a strong foundation for our study, we will first undertake a thorough review of the existing literature. This review will explore the evolution of methodologies and technologies used for big data analytics, tracing the journey from older approaches like MapReduce to the cutting-edge capabilities of modern frameworks like Spark. Following this, we will delve into the specifics of our methodology, detailing the process of selecting and implementing various statistical functions within the chosen framework. Subsequently, a rigorous performance analysis will be conducted to assess their effectiveness. The Discussion section will then serve as a platform to interpret the results, highlighting key findings and offering valuable insights into the strengths and limitations of these technologies for statistical analysis.

Our primary contribution lies in comprehensively analyzing the capabilities of two mature parallel processing technologies, Hadoop MapReduce and Apache Spark, for implementing a diverse range of

+ Corresponding author
E-mail address: amirkia.oskooei@intellica.net.

statistical functions on Big Data. By analyzing their capabilities and validating our findings through evaluating the performance of the superior technology in depth, this study aims to provide valuable guidance to companies and individuals, empowering them to select the most suitable framework for their specific statistical analysis needs within the big data landscape.

2. Literature Review

The vast amount of data generated by modern sensors and online interactions has created a new era of Big Data. This huge and varied collection of information offers many opportunities for companies and researchers, but its size and complexity make it difficult to analyze without special tools. Big Data Analytics (BDA) has permeated nearly every facet of modern life, revolutionizing industries ranging from accounting and agriculture to healthcare and finance. Its applications extend far and wide, offering deep insights into customer behavior, market trends, and operational efficiency [5] [6]. Several studies delve into the power of BDA, highlighting its applications and impact in specific fields. However, a common thread emerges: the crucial role of efficient and scalable data processing frameworks. MapReduce, a pioneering technology, paved the way, offering advantages like simplicity, fault tolerance, and scalability. Its batch processing prowess made it particularly efficient for large-scale data analysis. However, limitations arose, including a rigid paradigm, unsuitability for iterative algorithms and real-time streaming, and reliance on disk I/O, which hampered performance in certain scenarios [7-9]. Several optimization techniques have been proposed to address these issues, tackling job execution, data locality, resource management, MapReduce workflows, skewed data handling, and machine-learning based algorithms [10-14].

The arrival of Spark marked a significant shift. Studies comparing the two frameworks [15] [18] reveal how Spark addresses MapReduce's shortcomings. It offers a unified platform for both batch and real-time processing, efficiently handling iterative algorithms through in-memory capabilities and Resilient Distributed Datasets (RDDs). This empowers faster development and execution of tasks like machine learning using MLib library and graph processing using GraphX. Additionally, Spark's lineage-based recovery mechanism offers superior fault tolerance compared to MapReduce's task re-execution approach. Furthermore, its richer API with higher-level abstractions like RDDs, DataFrames, and SQL datasets simplifies and streamlines programming compared to MapReduce's low-level key-value pairs. Extensive investigations have compared their performance [16] [17] [19], revealing that while Spark generally excels, MapReduce might hold advantages in specific scenarios. For straightforward batch processing of large, uniform datasets with low latency concerns, MapReduce's simpler model and lower overhead can be efficient. Similarly, in cases with limited memory resources or older clusters, its disk-based processing might outperform Spark's in-memory approach. Additionally, highly specialized tasks or established MapReduce workflows may be better served by their respective tools. However, the overall trend favors Spark as the preferred choice for most big data processing tasks due to its flexibility, performance optimizations, and wider range of capabilities [20] [21].

This paper explores statistical analysis using Apache Spark's data parallelism, focusing on its broad applicability across various data-intensive fields. Unlike previous works on user interface testing [22], software engineering metrics [23], or geophysical data management [24], this research emphasizes Spark's ability to handle diverse statistical computations efficiently. It complements user behavior analysis studies [25-26], [34-35] by focusing on statistical methods and their parallel execution in Spark. This showcases the versatility of data parallelism in scientific computing, in contrast to infrastructure discussions [27].

In conclusion, while MapReduce laid the foundation for big data processing, its limitations prompted the evolution towards Spark. By offering greater flexibility, in-memory processing, and language support, Spark empowers data analysts to delve deeper into big data, extract more meaningful insights, and unlock the true potential of statistical analysis in the big data era. While existing big data research mentioned above explores various components and tools, it often overlooks the specific needs of statistical analysis. This study fills this gap by evaluating suitable frameworks for implementing and measuring the performance of various statistical functions (descriptive, exploratory, inferential) on Big Data. Our focus on statistical analysis within the big data landscape helps researchers and practitioners choose the most effective tools for their specific needs.

3. Methodology

To ensure a robust evaluation, this methodology section outlines a multi-step process. First, we meticulously select appropriate statistical functions for implementation. Next, we prepare the chosen dataset to ensure its compatibility with the chosen big data processing technology. Crucially, a comprehensive comparison is then conducted to identify the most suitable technology for our analysis needs. Finally, we establish a rigorous evaluation methodology to assess the performance of the chosen framework when executing the selected functions.

3.1. Identifying Functions and Preparing Data

The first step involved pinpointing the most frequently used and potent statistical functions that form the backbone of data analysis. We meticulously selected functions that fell into three distinct categories: descriptive (mean, median, variance, correlation, cumulative sum, mode), exploratory (min-max, percentile, histogram, skewness and kurtosis), and inferential (chi-square test, linear regression). This categorization ensured that our study encompassed a diverse range of statistical tools commonly employed across various disciplines. As for the dataset, we opted for the widely-used Iris dataset due to its advantageous properties for our experimental study. Firstly, its prevalence enables direct comparison with existing literature, facilitating assessment of our findings' generalizability and ensuring alignment with established practices. Its four numerical features (sepal and petal length/width) and three distinct classes (iris setosa, virginica, and versicolor) provide a fertile ground for exploratory data analysis and meaningful visualizations. Furthermore, while not perfectly balanced, the class sizes within the Iris dataset are close enough to enable valid statistical comparisons and hypothesis testing. To simulate the challenges of big data, we meticulously enlarged the Iris dataset by factors of 10,000 and 100,000, creating datasets of 1.5 million and 15 million rows respectively.

3.2. Framework Selection

Following a detailed examination and literature review, it was concluded that Apache Spark was the optimal framework. Our decision to leverage Apache Spark over MapReduce stemmed from a comprehensive evaluation of their strengths and weaknesses. MapReduce, characterized by its distributed processing paradigm, tackles the big data challenge by splitting data into manageable chunks ("maps") processed across multiple nodes. These individual "maps" then underwent a "reduce" phase where the results were aggregated, unlocking key benefits. Its parallel processing power across multiple nodes significantly reduces processing time. The distributed architecture ensures fault tolerance by automatically re-assigning tasks on failures, guaranteeing uninterrupted analysis despite hardware hiccups. Finally, its memory-conscious design, processing data in chunks and relying on disk storage, makes it efficient on systems with limited memory, broadening its accessibility [28] [29].

While MapReduce undeniably shines in some aspects, its limitations became significant roadblocks in our pursuit of in-depth statistical exploration. One of the key constraints of MapReduce is its rigid map-reduce programming paradigm. This approach, while effective for specific tasks, can be restrictive for expressing complex statistical logic inherent in many analytical models. This rigidity would have limited our ability to implement the nuanced statistical algorithms. Furthermore, MapReduce's reliance on disk storage for iterative algorithms creates a significant performance bottleneck. Since many statistical analyses involve iterative calculations, this reliance on disk I/O would have resulted in slower processing times. Finally, the low-level programming requirement of MapReduce, primarily in Java, presents a barrier to entry for data analysts who are more accustomed to higher-level languages like Python or R [30] [31].

In contrast, Apache Spark emerged as a more suitable framework due to its advancements that specifically address the limitations of MapReduce in statistical analysis. Spark's core strength lies in its ability to build upon the foundation of MapReduce while overcoming its shortcomings. One of the most significant advantages of Spark is its flexibility that extends beyond the map-reduce logic. Spark offers built-in libraries like Spark SQL and MLib, which provide a rich collection of pre-implemented statistical functions. This vast library empowers us to explore a wider range of sophisticated statistical methods. Additionally, Spark's in-memory processing capability tackles the performance bottleneck that plagued MapReduce's iterative algorithms. By storing intermediate data in memory, Spark significantly reduces disk I/O, leading to faster analysis within a

shorter timeframe. Finally, Spark's language versatility through APIs in Python, R, Java, and Scala fosters wider accessibility for data analysts [32] [33]. In conclusion, Spark's ability to overcome MapReduce's limitations in statistical analysis makes it the ideal framework for our study.

3.3. Performance Evaluation

To gauge the effectiveness of the chosen framework, we calculated speedup values by executing the selected functions on a single node and then scaling to clusters of 2 and 3 nodes. Execution time served as the primary performance metric, providing a clear measure of efficiency. Each function was meticulously run three times, and the average execution time was calculated to ensure higher reliability and mitigate the impact of potential outliers. To maintain system stability and ensure consistent results, we implemented stringent measures, ensuring the difference in execution time across runs remained consistently within 2%. This meticulous approach helped us draw reliable conclusions about the performance gains achieved through parallel processing. Furthermore, we leveraged the Spark Dashboard as a valuable tool to monitor job specifications, performance metrics, and resource utilization, providing real-time insights into the execution process.

4. Implementation and Results

To harness the power of Spark, we opted for Python (PySpark), considering its readability and popularity among data scientists. Our development environment was PyCharm IDE. Spark offers various data structures, and we chose DataFrames for their structured nature and high-level API. After loading the data into DataFrames, we applied the functions using Spark SQL and MLib libraries. Spark SQL handled functions like Mean, Median, Variance, and more, while MLib tackled Linear Regression, Chi-Squared Test, and Correlation.

Our experimental setup utilizes a single core of AMD Ryzen 5 2500U processor for each node in the cluster. As a recap, a Spark job comprises client and Spark sides. The client side houses the driver program (our code). On the Spark side, a cluster exists with a single master node (cluster manager) and worker nodes (containing executors that receive tasks). For this study, we employed the default Spark Standalone Cluster Manager. Each worker node was assigned 2GB of memory, with 1GB of overhead. We executed the code in three configurations: single node, 2-node cluster, and 3-node cluster.

The provided Table 1 showcases the calculated speedup values achieved when transitioning from a single node to clusters of 2 and 3 nodes.

Table 1. Execution time and Speedup values in 3 different configurations.

Function	Nodes	Dataset (433 MB)		Dataset (43 MB)	
		Time (s)	SpeedUp	Time (s)	SpeedUp
Mean	1	142.8	0%	20.3	0%
	2	92.2	35%	14.5	29%
	3	68.5	52%	13.0	36%
Median	1	169.0	0%	23.8	0%
	2	116.7	31%	15.9	33%
	3	81.8	52%	14.1	41%
Min-Max	1	141.0	0%	19.9	0%
	2	91.2	35%	14.7	26%
	3	70.0	50%	12.9	35%
Histogram	1	67.7	0%	14.2	0%
	2	45.7	32%	11.7	18%
	3	36.5	46%	11.1	22%

Cumulative Sum	1	136.7	0%	19.0	0%
	2	85.5	37%	14.3	25%
	3	64.4	53%	12.7	33%
Percentile	1	91.5	0%	14.2	0%
	2	55.7	39%	11.0	23%
	3	44.8	51%	10.4	27%
Correlation	1	417.4	0%	48.6	0%
	2	272.9	35%	32.7	33%
	3	187.1	55%	26.4	46%
Skewness and Kurtosis	1	151.0	0%	23.4	0%
	2	100.2	34%	17.4	26%
	3	75.7	50%	15.5	34%
Variance	1	141.5	0%	20.4	0%
	2	93.3	34%	14.8	28%
	3	69.5	51%	13.5	34%
Chi-Square test	1	61.8	0%	16.5	0%
	2	41.0	34%	12.4	24%
	3	33.6	46%	11.7	29%
Linear Regression	1	114.2	0%	21.9	0%
	2	86.4	24%	16.6	24%
	3	70.1	39%	14.9	32%
Mode	1	168.5	0%	29.1	0%
	2	113.8	32%	16.2	44%
	3	82.0	51%	13.7	53%
Count	1	21.7	0%	8.1	0%
	2	16.4	25%	7.5	8%
	3	13.5	38%	7.3	9%

Analyzing the bar charts provided in Figure 1 allows us to compare the average execution times of each function (in seconds) across different cluster configurations (single node, 2-node, and 3-node). Interestingly, a consistent pattern emerges for both datasets: correlation function exhibits the longest execution time by a significant margin, while count function consistently takes the least time. This disparity can be attributed to the inherent complexity of calculating correlations compared to the simpler counting operation. Other functions fall between these extremes, with their execution times influenced by factors such as the number of iterations required and the amount of data involved.

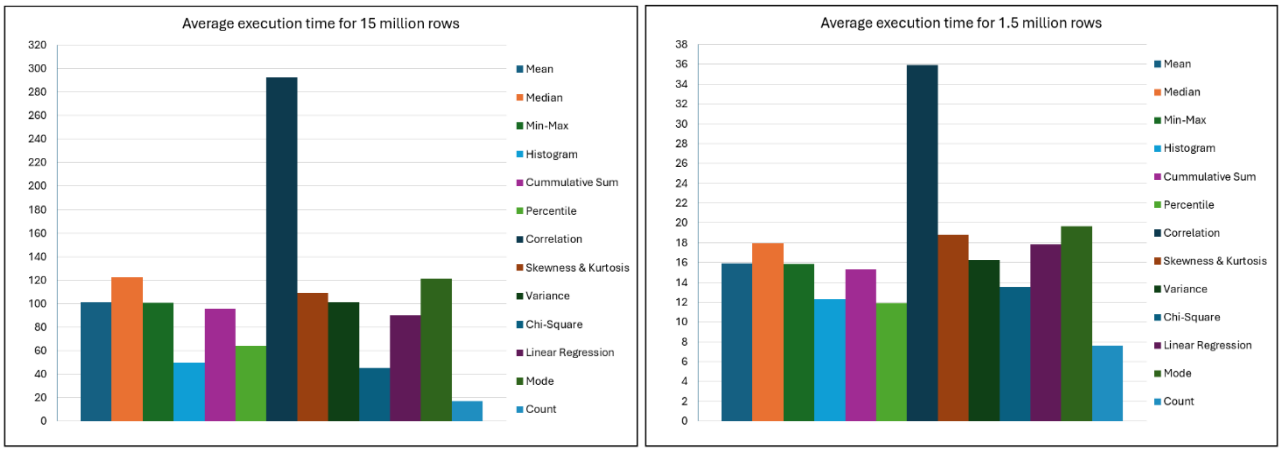

Fig. 1: Bar charts for comparing execution time of selected functions.

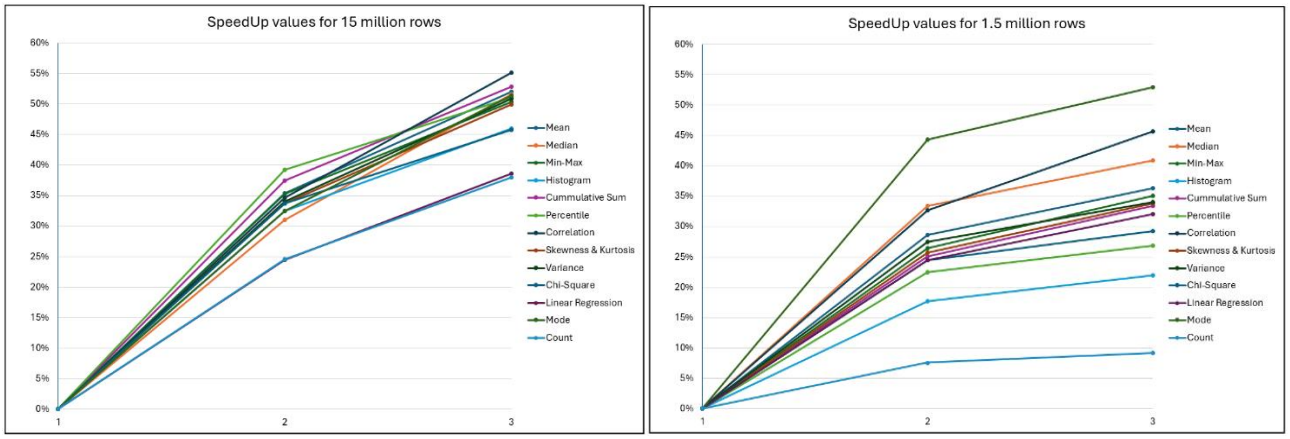

Fig. 2: Line charts for comparing Speedup values of selected functions.

The line charts in Figure 2 showing speed-up values (in percent) offer compelling insights into the performance benefits of parallel processing within Spark. As expected, transitioning from a single node to a 2-node cluster leads to significant improvements for all functions in both datasets. This observation underscores the effectiveness of Spark in leveraging multiple cores to accelerate computations. Notably, the jump from 1 to 2 nodes is more pronounced than the subsequent jump from 2 to 3 nodes, as evidenced by the steeper slopes in the lines. This suggests that adding the first additional node has the most significant impact, potentially due to factors like reduced data shuffling and improved task distribution within the cluster. Additionally, the dataset size influences the observed speedup values. While the smaller 1.5 million row dataset exhibits a wider spread (ranging from 8% to 53%), indicating varying degrees of benefit from parallelization across different functions, the larger 15 million row dataset consistently shows higher speedup values (24% to 55%) for all functions. This suggests that larger datasets tend to benefit more uniformly from the increased processing power offered by a cluster, due to amortizing the overhead associated with coordination and communication within Spark. These findings offer valuable insights into the impact of both dataset size and cluster configuration on the performance of various statistical functions within Spark.

5. Discussion

Spark's versatility surpasses MapReduce, particularly for statistical analysis. While it can efficiently utilize MapReduce logic for basic functions like mean or median, Spark excels when dealing with more complex tasks. For functions that involve shuffling, merging, or sorting – like calculating percentiles, skewness, or kurtosis – Spark's in-memory processing delivers substantial performance boosts. This advantage extends to iterative algorithms like linear regression, where Spark's efficiency shines. Overall, Spark's adaptability and ability to leverage in-memory processing solidify its position as the preferred framework for tackling a wider range of statistical analysis tasks on big data.

While our research has established Spark as the preferred framework for most statistical analysis tasks on big data, it is crucial to acknowledge scenarios where MapReduce might still hold an advantage. For simple, large-scale batch processing jobs, or when dealing with very large, static datasets residing in the HDFS (Hadoop Distributed File System), MapReduce can be a more cost-effective option. Given that the data is readily available in HDFS, the in-memory processing that Spark offers becomes unnecessary in such circumstances. This approach eliminates the overhead associated with managing Spark clusters, in-memory data structures, and data shuffling. Additionally, it reduces network traffic and avoids the need for expensive infrastructure required for in-memory processing of massive datasets. Therefore, when budget constraints are a significant concern, and the task aligns well with MapReduce's strengths, it remains a viable choice.

Our analysis of the results revealed a noteworthy trend: the speedup achieved when transitioning from 1 to 2 nodes was generally higher than the improvement observed from 2 to 3 nodes. This phenomenon can be attributed to several factors. As the number of nodes increases, communication and synchronization overhead can grow, potentially impacting performance gains. Additionally, imbalanced data distribution across nodes can lead to inefficiencies. Furthermore, Amdahl's Law states that the task's inherent sequential nature limits the potential speedup from parallelization. As more nodes are added, the benefits of further parallelization diminish, even for highly parallelizable tasks.

Our work focuses on the suitability of Spark for statistical analysis while evaluating its performance. Achieving speedup values of up to 55%, effectively halving the execution time, highlights the remarkable potential of Spark as an exceptionally efficient data parallelism and big data processing tool.

6. Conclusion and Future Works

This study investigated Apache Spark's effectiveness for Big Data statistical analysis. Spark's advantages in library offerings, high-level APIs, language support, in-memory processing, and flexibility were evident when compared to other frameworks. The performance evaluation further solidified these advantages, demonstrating significant improvements across various statistical functions and datasets. Our findings can be valuable for data analysts and programmers who need to choose the appropriate framework for their specific tasks. Additionally, businesses gain valuable insights into how each framework tackles the challenges of big data and the extent to which these issues can be mitigated.

7. Acknowledgement

We express our sincere gratitude to Intellica Consultancy for providing the essential environment and resources for this research.

8. References

[1] Kshetri, N.: The Emerging Role of Big Data in Key Development Issues: Opportunities, Challenges, and Concerns. 1, 2, (2014). https://doi.org/10.1177/2053951714564227.

[2] Wang, C. et al.: Statistical methods and computing for big data. 9, 4, (2016). https://doi.org/10.4310/SII.2016.V9.N4.A1.

[3] Targio Hashem, I.A. et al.: MapReduce: Review and open challenges. 109, 1, (2016). https://doi.org/10.1007/S11192-016-1945-Y.

[4] Salloum, S. et al.: Big data analytics on Apache Spark. 1, 3, (2016). https://doi.org/10.1007/S41060-016-0027-9.

[5] Rodríguez-Mazahua, L. et al.: A general perspective of Big Data: applications, tools, challenges and trends. 72, 8, (2016). https://doi.org/10.1007/S11227-015-1501-1.

[6] Furht, B., Villanustre, F.: Big data technologies and applications. (2016). https://doi.org/10.1007/978-3-319-44550-2.

[7] Li, R. et al.: MapReduce Parallel Programming Model: A State-of-the-Art Survey. 44, 4, (2016). https://doi.org/10.1007/S10766-015-0395-0.

[8] Doulkeridis, C., Nørvåg, K.: A survey of large-scale analytical query processing in MapReduce. (2013). https://doi.org/10.1007/s00778-013-0319-9.

[9] Malleswari, T.Y.J.N., Vadivu, G.: MapReduce: A technical review. Indian Journal of Science and Technology. 9, (2016). https://doi.org/10.17485/ijst/2016/v9i1/78964.

[10] Gu, R. et al.: SHadoop: Improving MapReduce performance by optimizing job execution mechanism in Hadoop clusters. 74, 3, (2014). https://doi.org/10.1016/J.JPDC.2013.10.003.

[11] Lee, S. et al.: Performance Improvement of MapReduce Process by Promoting Deep Data Locality. October 1 (2016). https://doi.org/10.1109/DSAA.2016.38.

[12] Palanisamy, B., Singh, A., Liu, L.: Cost-Effective Resource Provisioning for MapReduce in a Cloud. (2015). https://doi.org/10.1109/tpds.2014.2320498.

[13] Li, W., Tang, M.: The performance optimization of big data processing by adaptive MapReduce workflow. IEEE Access. 10, 79004–79020 (2022). https://doi.org/10.1109/access.2022.3193770.

[14] Irandoost, M.A. et al.: MapReduce Data Skewness Handling: A Systematic Literature Review. 47, 5, (2019). https://doi.org/10.1007/S10766-019-00627-0.

[15] Gopalani, S., Arora, R.R.: Comparing Apache Spark and Map Reduce with Performance Analysis using K-Means. (2015). https://doi.org/10.5120/19788-0531.

[16] Benlachmi, Y., Hasnaoui, M.L.: Big data and Spark: Comparison with Hadoop. July 27 (2020). https://doi.org/10.1109/WORLDS450073.2020.9210353.

[17] Vishal, A. et al.: Performance comparision of Hadoop and spark engine. February 1 (2017). https://doi.org/10.1109/I-SMAC.2017.8058263.

[18] Singh, A. et al.: Performance comparison of Apache Hadoop and Apache Spark. June 15 (2019). https://doi.org/10.1145/3339311.3339329.

[19] Ibtisum, S. et al.: A comparative analysis of big data processing paradigms: Mapreduce vs. apache spark. (2023). https://doi.org/10.30574/wjarr.2023.20.1.2174.

[20] Benlachmi, Y., Hasnaoui, M.L.: Big data and Spark: Comparison with Hadoop. July 27 (2020). https://doi.org/10.1109/WORLDS450073.2020.9210353.

[21] Maheshwar, R.C., Haritha, D.: Survey on high performance analytics of bigdata with apache spark. (2016). https://doi.org/10.1109/icaccct.2016.7831734.

[22] Uygun, Y. et al.: On the Large-scale Graph Data Processing for User Interface Testing in Big Data Science Projects. December 10 (2020). https://doi.org/10.1109/BIGDATA50022.2020.9378153.

[23] Kapdan, M. et al.: On the Structural Code Clone Detection Problem: A Survey and Software Metric Based Approach. June 30 (2014). https://doi.org/10.1007/978-3-319-09156-3_35.

[24] Pierce, M. et al.: The QuakeSim Project: Web Services for Managing Geophysical Data and Applications. 165, 3, (2008). https://doi.org/10.1007/S00024-008-0319-7.

[25] Olmezogullari, E., Aktas, M.S.: Pattern2Vec: Representation of clickstream data sequences for learning user navigational behavior. (2021). https://doi.org/10.1002/CPE.6546.

[26] Olmezogullari, E., Aktas, M.S.: Representation of Click-Stream DataSequences for Learning User Navigational Behavior by Using Embeddings. Presented at the December 10 (2020). https://doi.org/10.1109/BIGDATA50022.2020.9378437.

[27] Nacar, M.A. et al. (2007), VLab: collaborative Grid services and portals to support computational material science. Concurrency Computat.: Pract. Exper., 19: 1717-1728. https://doi.org/10.1002/cpe.1199.

[28] Dean, J.M., Ghemawat, S.: MapReduce: Simplified data processing on large cluster. (2018). https://doi.org/10.21276/ijre.2018.5.5.4.

[29] Lee, K.-H. et al.: Parallel data processing with MapReduce: a survey. 40, 4, (2012). https://doi.org/10.1145/2094114.2094118.

[30] Weets, J.-F. et al.: Limitations and challenges of HDFS and MapReduce. Presented at the October 8 (2015). https://doi.org/10.1109/ICGCIOT.2015.7380524.

[31] Abaker Targio Hashem, I. et al.: MapReduce: Review and open challenges. 109, 1, (2016). https://doi.org/10.1007/S11192-016-1945-Y.

[32] Salloum, Salman, et al. "Big data analytics on Apache Spark." International Journal of Data Science and Analytics 1 (2016): 145-164.

[33] Shi, J. et al.: Clash of the titans: MapReduce vs. Spark for large scale data analytics. 8, 13, (2015). https://doi.org/10.14778/2831360.2831365.

[34] M. Oz, et al. “On the Use of Generative Deep Learning Approaches for Generating Hidden Test Scripts”, International Journal of Software Engineering and Knowledge Engineering, Vol 31, Issue 10, p1447, 2021.

[35] I. Erdem, et al. “Test Script Generation Based on Hidden Markov Models Learning From User Browsing Behaviors”, 2021 IEEE International Conference on Big Data (Big Data), IEEE Computer Society, 2021.