A Big Data Optimization Comparison using Apache Spark and Apache Airflow

Conference Paper · January 2025

DOI: 10.1109/CCWC62904.2025.10903735

6 authors, including:

Samir Poudel
Middle Tennessee State University
16 PUBLICATIONS **46** CITATIONS
SEE PROFILE

Jiblal Upadhya
Middle Tennessee State University
10 PUBLICATIONS **21** CITATIONS
SEE PROFILE

Md Nahid Hasan
Middle Tennessee State University
7 PUBLICATIONS **15** CITATIONS
SEE PROFILE

Kritagya Upadhyay
Middle Tennessee State University
30 PUBLICATIONS **273** CITATIONS
SEE PROFILE

A Big Data Optimization Comparison using Apache Spark and Apache Airflow

Taylor Hartman*, Samir Poudel†
Jiblal Upadhya†, Md Nahid Hasan†, Kritagya Upadhyay*, Khem Poudel*
Department of Computer Science*, Computational and Data Science†
Middle Tennessee State University
Murfreesboro, TN, 37132
Email: {ju2i, tah6k, sp2ai, mh2ay}@mtmail.mtsu.edu,
{kritagya.upadhyay, khem.poudel}@mtsu.edu

2025 IEEE 15th Annual Computing and Communication Workshop and Conference (CCWC) | 979-8-3315-0769-5/25/$31.00 ©2025 IEEE | DOI: 10.1109/CCWC62904.2025.10903735

***Abstract*—While comparisons between Apache Hadoop and Apache Spark are well-documented, there has been limited research comparing Apache Spark with Apache Airflow, especially in terms of speed and memory usage. With Apache Airflow's recent introduction of dynamic task mapping, which performs similar functions to Apache Spark's map operation, a detailed comparison between the two tools has become increasingly relevant. A comparison in these areas would provide valuable insights for the Big Data Science community, helping determine which methods are better suited for tasks requiring high speed and efficient memory usage. This study focuses on comparing the Apache Spark Map function and Apache Airflow Dynamic Task Mapping function on two key metrics: memory utilization and computation speed. Specifically, we evaluate their performance in sorting formatted electrocardiogram sensory data. We hypothesize that Apache Spark will demonstrate faster processing times due to its advanced in-memory processing and sorting algorithms. However, this speed advantage is expected to come with higher memory usage compared to Apache Airflow. Our findings provide actionable insights into the strengths and limitations of these tools, guiding data scientists and engineers in choosing the most suitable framework for specific big data processing tasks. These results are particularly relevant for large-scale data sorting and transformation operations, contributing to informed decision-making in the Big Data Science community.**

***Index Terms*—Apache Airflow, Apache Spark, Big Data, Hadoop, Parallel Processing**

I. Introduction

A question has presented itself within the last few years of how we can make use of Big Data in healthcare applications. With multiple repositories of patient and electronic sensory data, we have entered the Big Data era. Big Data is known as data that together are too large to be stored, managed, analyzed, and captured in typical databases. [1] Previously, systems would take the data from the patient, doctor, or healthcare institution and load it directly into a relational database without previously harmonizing, pre-processing, and polishing the data. [2] This leads to an over-saturation of data without a specific intent for use of the data or being data rich but information poor. The healthcare industry has been generating and collecting vast amounts of data from various sources, including electronic health records, medical imaging, genomics, wearable devices, and social media. This data is referred to as Big Data, and it presents both challenges and opportunities for healthcare providers and researchers.

Effectively preprocessing data from diverse sources and formats is a significant challenge in big data analytics. Data often originates from various countries, databases, and query structures, necessitating robust solutions for integration and transformation. For instance, the heterogeneity of data sources can impede data visualization and prediction, affecting analytical results accordingly [2], [3]. Addressing these challenges requires frameworks capable of near-real-time or real-time data transformation to facilitate accurate analytics and predictions.

Technologies such as Apache Hadoop, Apache Spark, and Apache Airflow have emerged to manage large-scale, heterogeneous datasets. Apache Hadoop provides a distributed storage and processing framework, enabling efficient handling of vast data volumes [4], [5]. Building upon this, Apache Spark offers in-memory computing capabilities, enhancing the speed of iterative data processing tasks [6]. Apache Airflow, with its workflow orchestration features, allows for the scheduling and monitoring of complex data pipelines, facilitating dynamic task mapping and real-time data integration [7]. These frameworks represent a shift toward more intelligent data processing tools that prioritize efficiency and flexibility in handling diverse data sources [8].

In healthcare, the application of these technologies holds significant promise. Advanced data preprocessing and real-time analytics can enable early detection of diseases such as cancer, improving treatment outcomes [9], [10]. Predictive models can also identify patients likely to miss appointments, allowing healthcare providers to optimize scheduling and enhance access to care [11]. By integrating diverse datasets—from electronic health records to sensor-generated data—these frameworks support the development of systems that enhance operational efficiency and patient outcomes. As the demand for intelligent, real-time decision-making in healthcare grows, the role of robust big data frameworks becomes increasingly critical.

The use of Big Data in healthcare has the potential to transform the industry by improving patient outcomes, advancing research efforts, and reducing costs. One of the primary benefits of Big Data is the potential to enable personalized

and precision medicine [12]. By analyzing large data sets, healthcare providers can identify patterns and correlations that may not be apparent in smaller data sets, leading to more accurate diagnoses and treatment plans tailored to individual patients' needs. Big Data can also enable real-time monitoring of patient health, providing opportunities for early intervention and prevention of adverse health outcomes. Furthermore, Big Data can support research efforts by providing access to large and diverse data sets, enabling researchers to test hypotheses and develop new insights that can lead to scientific breakthroughs. For example, Big Data analytics can be used to identify new disease markers or genetic mutations that contribute to the development of diseases. Despite the potential benefits of Big Data, there are also challenges that need to be addressed, such as privacy concerns, data security, and data quality. Additionally, there is a need for skilled professionals who can effectively manage and analyze large data sets in the healthcare industry. Optimizing Big Data processing workflows is crucial to improve performance and reduce processing time in healthcare applications. The size of the data set, the complexity of the workflow, and the computational resources available are some of the factors that can impact the performance of Big Data processing workflows. Big Data analytics has immense potential to transform healthcare by enabling personalized and precision medicine and advancing research efforts. However, it is crucial to address the challenges associated with Big Data, including privacy concerns, data security, and data quality, to realize its full potential. Optimizing Big Data processing workflows can further improve performance and reduce processing time in healthcare applications.

II. Background

A. Apache Hadoop

Apache Hadoop is self-described as a framework that allows for the distributed process of large datasets across clusters of computers using simple programming models [13]. Hadoop implements Google's MapReduce approach using the same of hiding complexity from users so they can focus on programming. This framework provides store, manipulate, and extract information from Big Data in several ways [14]. With Apache Hadoop being a popular open source and useful software framework that enables distributed storage and storing big data across clusters, it developed in a way that can scale up from a single server to thousands of nodes. The core parts of Hadoop are Hadoop Distributed File System (HDFS) and MapReduce [15]. HDFS chunks the data into small blocks and saves them into different nodes. MapRedce is a computing framework. The MapReduce processing is slow due to read and write from the disk [16] . To do this with Big Data that was too large to be used in traditional databases, Hadoop uses Hadoop Distributed File System (HDFS) which is a fault-tolerant data storage file system. In our research, similar to HDFS data processing, we are interested in looking into the CPU time and memory used in similar Big Data technology like Apache Spark [17] and Apache Airflow.

B. Apache Spark

Apache Spark is self-described as a "Unified engine for large scale data analytics [18]." This framework is built on top of Hadoop but with some internal changes. Spark has many built-in features that make it ideal for big data analytics [19]. There are roughly 80 high-level operators associated with parallel processing as well as SQL, Streaming (data streaming), MLlib (machine learning), and GraphX (graphs) libraries [20], [21]. These features can work together in unison to create a great computing infrastructure for machine learning and big data processing. Spark provides many advantages for developers to build big data applications. There are two important terminology in spark: Resilient Distributed Datasets (RDD) and Directed Acyclic Graph (DAG). Those two techniques can make the speed faster up to ten times as compared to Hadoop. Spark uses memory and the MapReduceing process of spark is faster as compare the Hadoop. Spark bench-marking suite improves the optimization [22] of workload configuration [23]. Spark can handle multiple source data with cached support and perform parallel operation using fault tolerance mechanism [16]. The speed and memory performance of Hadoop and Spark significantly depends on the parameter's configurations such as word load, data size, and cluster architecture. Comparisons of Apache Spark and Apache Hadoop are well documented, however, as of the time of this publication, there are no known documented comparisons of Apache Spark's MapReduce and the newly introduced Apache Airflow equivalent.

C. Apache Airflow

Apache Airflow is an "open-source platform for developing, scheduling, and monitoring batch-oriented workflows" [24]. Traditionally it has been used as a data workflow manager. Airflow has a built-ion scheduler that allows a user to schedule reoccurring data workflows allowing automation to occur. This automation is somewhat fault tolerant due to Airflow having built in semantics that allows a task to be executed without the previous task scheduled within the data pipeline to have executed successfully. Apache Airflow is very modular and offers many pre-built interfaces that can connect common cloud databases. Airflow supports containerized execution and Kubernetes operator.

D. Previous Work

1) Previous Comparisons: Previous work has shown a technology called Apache Spark compared against the likes of Apache Hadoop in these specific categories. The data found that Spark processed data at a rate 100 times faster than Hadoop's MapReduce function and the memory management is less than Hadoop because it performs the operations in a "lazy" fashion, which is to say an operation takes place after the transformation of the data happens, and because the operation takes place in cache memory, but this takes place in near real-time only [25].

2) Apache Airflow's Dynamic Task Mapping: Traditionally, Apache Airflow has been widely utilized as a data flow manager and scheduler, orchestrating complex workflows across distributed systems. Recently, however, a new feature—Dynamic Task Mapping—has been introduced, which enhances Airflow's ability to handle data processing tasks in real time. This feature leverages Airflow's existing strengths in workflow management while integrating functionality reminiscent of Hadoop's map and reduce operations, allowing it to dynamically map tasks to data as it arrives, enabling more efficient and flexible processing capabilities [11].

To date, there are no known studies that directly compare Apache Spark's Map function with Apache Airflow's Dynamic Task Mapping feature. This lack of comparative analysis represents a critical gap in the literature, particularly given the increasing demand for optimized big data processing solutions that balance speed and memory efficiency.

III. Method

A. Dataset

The dataset used in this study is a structured .csv file containing electrocardiogram (ECG) data, organized with 140 columns and 4,998 rows, where each row represents a complete ECG recording for an individual patient [26]. Each ECG record consists of 140 sequential data points, captured as floating-point numbers, which reflect the electrical activity at various moments in the heartbeat cycle. Columns 0 to 139 store these data points, providing a detailed temporal profile of each patient's ECG.

The dataset includes a categorical label in a separate column, indicating whether the ECG recording is classified as "normal" or "abnormal." This label is represented as a binary variable with values of either 0 or 1, though for the purposes of this experiment, the specific mapping of 0 or 1 to normal or abnormal is not necessary. This streamlined representation of ECG data facilitates efficient processing and analysis, making it suitable for the performance benchmarking tasks in this study.

B. Computing Comparison Metrics

The Python language is used for finding and computing all comparison metrics. In our case, the metrics are the memory used during a specific process and the time it takes for the transformation of data to occur. All libraries and methods discussed below were used in both Apache Spark and Apache Airflow to find, compute, and extract the metrics for analysis.

To monitor and process the delta time of a process, the method **datetime.datetime.now()** was used from the library **datetime**. This method returns the **HH:MM:SS** of the current time once the object is called. So far, this object returns time accurate to the microsecond (10^{-6}). To find the delta time, **timedelta** was not used in this experiment. This was done to ensure proper computing of the delta time of only the transformation process of the data by being able to select specific start and end time positions. Variables start_time and end_time were used with start_time being set to **datetime.datetime.now()** at the beginning of the transformation. The end_time was calculated by calling **datetime.datetime.now()** and subtracting start_time from it to find the delta time of the action taken on the data. The method **.total_seconds()** was used to convert the returned datatime object into an easier to read form of only seconds.

To identify the specific process of the running code block the **os** and **psutil** libraries were used. The process was found by calling the method **.getpid()** of the **os** library which returns the integer designation of the current process. This was integer passed inline to the **psutil** method **.Process()** to get a list of process specific information. We use this object and call the method **.memory_info().rss()** which returns the bytes used for the process. This number is divided by 10^6 to give the megabytes used within the specified process.

C. Experiment Data Flow

The data flow map is shown below in Figure 2 to illustrate the comparison model. The first step in our experiment was to take electrocardiogram (ECG) data [26] from the source and download it to the local machine to be utilized throughout this experiment. The machine is a MSI laptop with a 9th GEN INTEL® CORE™i7-9750H processor, NVIDIA GEFORCE® RTX™ 2070 MAX-Q, with 16GB DDR4 system memory. Apache Spark and Apache Airflow were installed with the necessary dependencies on the machine locally using the Linux System Ubuntu for Windows 11. Once installation was complete and confirmed through code testing, the process of identifying the code required to accomplish our task started. Once these python files were complete the transformation took place.

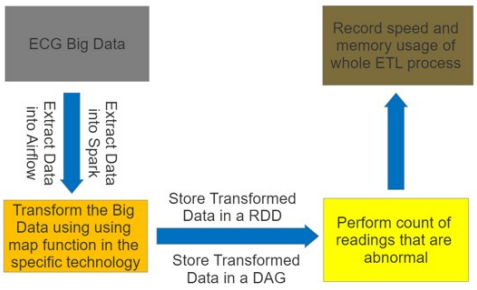

Fig. 1. Block Diagram showing the data flow of the comparison process.

D. Apache Spark

In Apache Spark, the conversion of a DataFrame to a Resilient Distributed Dataset (RDD) is a necessary step for certain operations, as the map function is specifically designed to be applied on objects of data type RDD. This conversion is performed using the .rdd method, which transforms the Spark DataFrame into an RDD to allow for more flexible and lower-level transformations inherent to Apache Spark's data processing architecture [27].

For this experiment, 1,000 individual runs were executed in a loop, with each run designed to release memory resources

back to the system. To ensure efficient memory management, the Spark session is explicitly terminated at the end of each loop iteration using spark.stop() and re-initialized at the start of the next iteration via a constructor call. This procedure allows Spark to release memory after each run, preventing resource bottlenecks and enhancing performance consistency across iterations.

The transformation algorithm applied to the RDD utilizes an inline lambda function defined as lambda x: (x[0:-2], float(x[-1])). This function is embedded within the .map() method to process the raw data RDD, selectively transforming each data row by extracting specific elements. The resultant RDD is then stored in a variable, enabling further analysis and facilitating later stages of data processing. This approach leverages Spark's distributed data handling capabilities, allowing for efficient and scalable data transformations.

E. Machine Learning

The test data set was then transformed further, utilizing Apache Spark, to assemble a RDD that was compatible with Spark Machine Learning algorithms. This data frame was separated into two columns. One with the features needed for machine learning algorithms and the other as the desiccators of 1 or 0 depending on if those ECG readings are normal or abnormal, respectfully. This RDD was then passed to the following Spark Machine Learning (Spark MLlib):

- Random Forrest
- Gradient-Boosted Tree
- Multinomial Logistic Regression
- Binomial Logistic Regression

IV. Results and Discussion

A. Apache Spark

During our research testing we ran the Apache Spark code 1000 times (n), saved the time and memory data in an external file, and plotted them graphically. The data shows a normal distribution for the time it took to load and transform the data with a mean value of **0.1776305** seconds with a confidence interval of over 99%. The interesting result is in the memory usage. The graph on the right in figure 3 shows the memory usage per iterations of the DAG run. The iterations are arranged chronologically from left to right. This shows an interesting trend in the distribution of the memory used for the experiment.

The first iteration utilized the least amount of memory through the n^{th} amount of runs. (Figure 4) The next 11 were of a higher memory usage. The middle third of the runs used the most memory out of all of the iterations. So far, this data makes sense assuming there could be a memory leak clogging up the cache data and not releasing all of the data back to the system upon stopping the Spark session. The median memory used was **89.432064** MB.

However, the bulk of the runs happens at a significantly less memory usage, almost a third less compared to the highest memory used. This could indicate that the Spark process "learned" in some way how to most efficiently handle

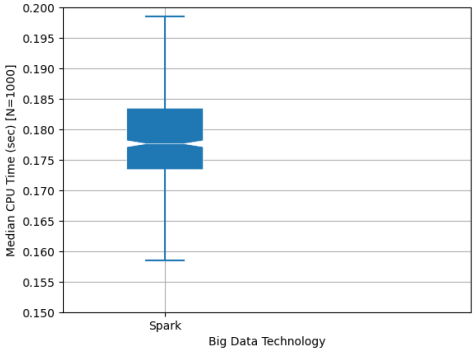

Fig. 2. Run times of Apache Spark

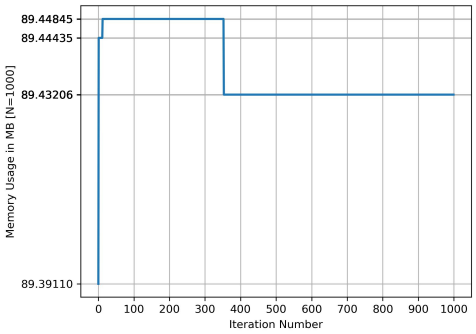

Fig. 3. Run times of Apache Spark

the loading and transforming of the data. There is no solid evidence to corroborate this claim. Further research into this matter would be needed to find out this answer.

B. Apache Airflow

While conducting research into Apache Airflow we found the following results from testing time and memory usage of loading in data. We were not able to perform the transformation of the data with the Dynamic Task Mapping due to the installed version of Apache Airflow being of a version that does not support this feature. We were unable to install the required version by the submission deadline to produce meaningful results. However, we were able to get results that indicate our assumptions of the technology in reference to the time and memory used will be correct.

This setback aside, the data collected for both the time and memory used during n runs was evenly distributed and easily analyzed. The median time it took to perform the actions was **0.1666595** seconds with a confidence interval of over 99% when performing only the load action of the experiment. The median memory used, which is also from only the load action, was **140.742656** MB with a confidence interval of the graphs in figure 4 visually represent this data.

C. Comparison

Apache Spark is considerably slower at loading and transforming data compared to Apache Airflow only loading in the date. The difference seems to be of a large enough amount,

TABLE I
APACHE SPARK MEMORY USAGE BY ITERATION

Memory Used	Number of Iterations
89.39110	1
89.44435	11
89.44845	341
89.43206	647

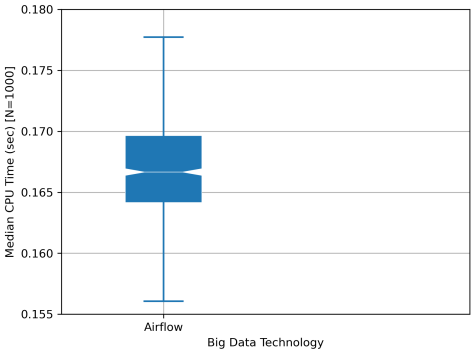

Fig. 4. Run times of Apache Airflow

given our experience, to reasonably predict that Apache Spark would still be slower compared to Apache Airflow. This takes into account the assumption that the completion of the experiment would have Apache Airflow transforming the data with the Dynamic Task Mapping function. (Figure 5 & Table 2)

TABLE II
MEDIAN TIMES FROM APACHE SPARK AND APACHE AIRFLOW

	Apache Spark	Apache Airflow
Median Time	0.1776305s	0.1666595s
Standard Error	0.000253s	0.000188s
Confidence Interval	0.000497s	0.000368s
Median Memory	89.432064	140.742656
Standard Error	0.00025	0.062157
Confidence Interval	0.000368	0.121828

With respect to the memory used, Apache Spark used considerably less memory to load in and transform the data using the **.map()** method. This evidence allows us to reasonably predict that if Apache Airflow were to transform the data with the Dynamic Task Mapping functionality, the memory used by Apache Airflow (table 2) would grow, therefore Apache Spark would still be using less memory overall.

D. Machine Learning

We found that the Binomial Logistic Regression was the most accurate model out of the models that performed correctly. (Figure 8) Two of the four models showed a final accuracy of 100% which is an incorrect assessment of a working model. This could be due to over fitting of the training data verses the testing data. Further investigation is needed to uncover the reason for this error.

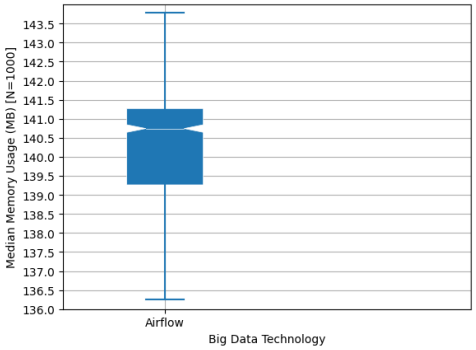

Fig. 5. Memory Usage of Apache Airflow

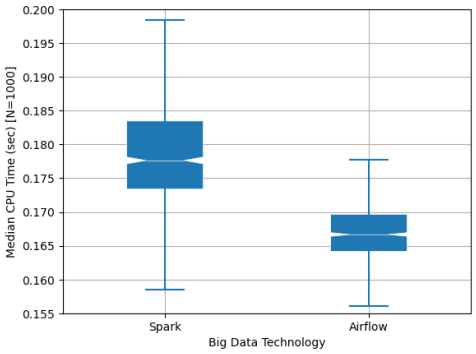

Fig. 6. Time data from Apache Spark and Apache Airflow

V. CONCLUSION

In conclusion, this study provided a comparative analysis of memory usage and CPU time between Apache Spark and Apache Airflow, two leading technologies in big data processing. Based on the current data and the assumption that the volume of data processed by Apache Airflow increases during execution, Apache Airflow demonstrates slightly faster

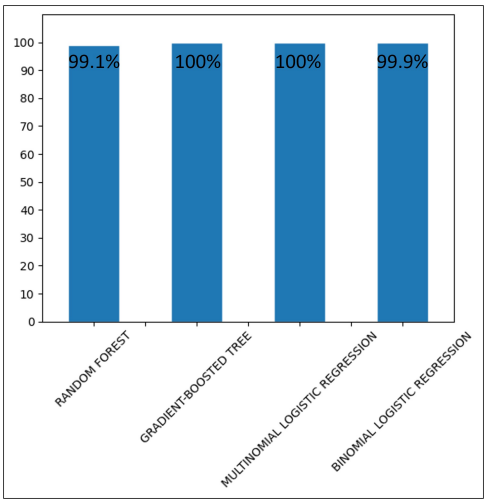

Fig. 7. Bar graph showing the machine learning algorithms and their accuracy's in percentage.

median processing times than Apache Spark, the latter provides a more memory-efficient solution, consistently maintaining lower memory usage across iterations. These findings suggest that Apache Spark may be better suited for memory-constrained environments, whereas Apache Airflow could be advantageous when minor speed gains are prioritized over memory efficiency.

These results are significant for the Big Data Science community, as they highlight the trade-offs between speed and memory efficiency when selecting a framework for specific applications. Understanding these differences is crucial for designing data flow pipelines tailored to various use cases, ensuring that the chosen technology aligns with the performance and resource requirements of the task at hand.

REFERENCES

[1] R. Pastorino, C. De Vito, G. Migliara, K. Glocker, I. Binenbaum, W. Ricciardi, and S. Boccia, "Benefits and challenges of big data in healthcare: an overview of the european initiatives," vol. 29, pp. 23–27.

[2] K. Alexakis, P. Kapsalis, Z. Mylona, G. Kormpakis, E. Karakolis, C. Ntanos, and D. Askounis, "Intelligent querying for implementing building aggregation pipelines," in *2022 13th International Conference on Information, Intelligence, Systems & Applications (IISA)*, pp. 1–6, IEEE.

[3] W. Raghupathi and V. Raghupathi, "Big data analytics in healthcare: promise and potential," *Health information science and systems*, vol. 2, no. 1, pp. 1–10, 2014.

[4] S. Salloum, R. Dautov, X. Chen, P. X. Peng, and J. Z. Huang, "Big data analytics on apache spark," *International Journal of Data Science and Analytics*, vol. 1, pp. 145–164, 2016.

[5] S. Tang, B. He, C. Yu, Y. Li, and K. Li, "A survey on spark ecosystem: Big data processing infrastructure, machine learning, and applications," *IEEE Transactions on Knowledge and Data Engineering*, vol. 34, no. 1, pp. 71–91, 2020.

[6] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin, *et al.*, "Apache spark: a unified engine for big data processing," *Communications of the ACM*, vol. 59, no. 11, pp. 56–65, 2016.

[7] S. Sakr, "Big data processing systems: State-of-the-art and open challenges," *2016 IEEE International Conference on Cloud Computing and Big Data Analysis (ICCCBDA)*, pp. 77–84, 2016.

[8] S. Poudel, Movinuddin, S. Gutta, R. K. Kommu, J. Upadhyay, M. N. Hasan, and K. Poudel, "Credit card batch processing in banking system," in *Proceedings of the Second International Conference on Advances in Computing Research (ACR'24)* (K. Daimi and A. Al Sadoon, eds.), (Cham), pp. 83–96, Springer Nature Switzerland, 2024.

[9] Y. Wang, L. Kung, and T. A. Byrd, "Big data analytics in healthcare: deep learning-based feature selection for electronic health record classification," *International Journal of Information Management*, vol. 43, pp. 328–341, 2018.

[10] T. Nhan, J. Upadhya, S. Poudel, S. Wagle, and K. Poudel, "Scalable multimodal machine learning for cervical cancer detection," in *2024 IEEE World AI IoT Congress (AIIoT)*, pp. 502–510, 2024.

[11] S. Dash, S. K. Shakyawar, M. Sharma, and S. Kaushik, "Big data in healthcare: management, analysis and future prospects," *Journal of Big Data*, vol. 6, no. 1, pp. 1–25, 2019.

[12] E. Nazari, M. H. Shahriari, and H. Tabesh, "Bigdata analysis in healthcare: apache hadoop, apache spark and apache flink," *Frontiers in Health Informatics*, vol. 8, no. 1, p. 14, 2019.

[13] "Apache hadoop."

[14] I. Polato, R. Ré, A. Goldman, and F. Kon, "A comprehensive view of hadoop research—a systematic literature review," vol. 46, pp. 1–25.

[15] T. Ivanov, A. Ghazal, A. Crolotte, P. Kostamaa, and Y. Ghazal, "CoreBigBench: Benchmarking big data core operations," in *Proceedings of the workshop on Testing Database Systems*, DBTest '20, pp. 1–6, Association for Computing Machinery.

[16] H. Ahmadvand, M. Goudarzi, and F. Foroutan, "Gapprox: using gallup approach for approximation in big data processing," vol. 6, no. 1, p. 20.

[17] Anonymous, "Apache spark: A big data processing engine," in *IEEE International Conference on Big Data*, pp. 800–805, IEEE, 2020.

[18] "Apache spark™ - unified engine for large-scale data analytics."

[19] J. G. Shanahan and L. Dai, "Large scale distributed data science using apache spark," in *Proceedings of the 21st ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, (Sydney, NSW, Australia), pp. 2323–2324, ACM, 2015.

[20] S. Alotaibi, R. Mehmood, I. Katib, O. Rana, and A. Albeshri, "Sehaa: A big data analytics tool for healthcare symptoms and diseases detection using twitter, apache spark, and machine learning," vol. 10, no. 4, p. 1398. Number: 4 Publisher: Multidisciplinary Digital Publishing Institute.

[21] H. Karau and R. Warren, *High performance Spark: best practices for scaling and optimizing Apache Spark*. " O'Reilly Media, Inc.", 2017.

[22] Anonymous, "Optimizing data processing: A comparative study of big data platforms in edge, fog, and cloud layers," *IEEE Transactions on Big Data*, 2020.

[23] N. Ahmed, A. L. C. Barczak, T. Susnjak, and M. A. Rashid, "A comprehensive performance analysis of apache hadoop and apache spark for large scale data sets using HiBench," vol. 7, no. 1, p. 110.

[24] "What is airflow? — airflow documentation."

[25] E. Nazari, M. H. Shahriari, and H. Tabesh, "BigData analysis in healthcare: Apache hadoop , apache spark and apache flink," vol. 8, no. 1, p. 14. Number: 1.

[26] "ECG dataset."

[27] M. Armbrust, R. S. Xin, C. Lian, Y. Huai, D. Liu, J. K. Bradley, X. Meng, T. Kaftan, M. J. Franklin, A. Ghodsi, *et al.*, "Spark sql: Relational data processing in spark," in *Proceedings of the 2015 ACM SIGMOD international conference on management of data*, pp. 1383–1394, 2015.