King Saud University

College of Computer and Information Science

Department of Software Engineering

Cloud Computing & Big Data (486 SWE)

SPARK VERSUS FLINK
Evaluation and Comparison

Layan Aluwaishiq 442201814
Nouf Abuabat 443200968
Saadiya Abdulqader 442204801
Sarah Alshenaifi 443200652
Sultanah Almutairi 443200444

Supervised By:
Dr.Sadeem Alsudais

Spark versus Flink: Evaluation and Comparison

Abstract

This study conducts a comprehensive evaluation and comparison of two prominent distributed data processing frameworks, Apache Spark and Apache Flink, leveraging the CC-News dataset, which comprises over 10 million Arabic news articles. The research assesses the efficiency of each framework in handling both batch and stream processing tasks within local and cloud-based environments. Key metrics such as resource utilization, latency, and throughput are examined to provide a holistic view of the frameworks' performance. Apache Spark demonstrates superior performance in batch processing, exhibiting high data transfer rates and scalability, whereas Apache Flink excels in stream processing, offering low latency and real-time processing capabilities. The findings highlight the importance of selecting the appropriate framework based on specific use cases, with Spark being more suitable for historical data analysis and Flink being ideal for real-time applications. This comparative analysis offers actionable insights into the strengths and limitations of each framework, facilitating informed decision-making in the context of big data applications.

1. Introduction

With the rapid advancements in technology and the growing reliance on digital solutions, big data has become a fundamental driver of progress across various sectors. The ongoing digital transformation in industries has amplified the significance of big data. As a result, efficient data processing and analysis have become crucial for optimizing business performance, fostering innovation, and contributing to this development.

2. Background

2.1. Big Data

Big data refers to large, complex datasets that are challenging to process using traditional methods. It is defined by three key aspects: **volume** (size), **variety** (different data types), and **velocity** (speed of generation). Technologies like Hadoop and Spark help analyze and extract insights from big data [18].

2.2. Cloud Computing

Cloud computing delivers computing resources (e.g., storage, servers) over the internet, offering scalability and cost efficiency. Major models include **IaaS** (infrastructure), **PaaS** (platforms), and **SaaS** (software)[16].

2.3. Distributed Data Processing

Distributed data processing involves splitting large tasks across multiple machines to speed up computation and increase reliability. Frameworks like **Hadoop** and **Spark** enable scalable, parallel processing of big data [19].

3. Dataset

Hugging Face CC-News Dataset (Arabic)

In this study, we aimed to identify a dataset meeting the criteria of containing at least 10 million Arabic text records, along with images and timestamps. After an extensive search across various data sources, including the Open Data Platform [29] and Kaggle[28], we identified a suitable dataset on Hugging Face[27]. This dataset, a subset of the CC-News collection, is licensed under Creative Commons Attribution[26],[7] and encompasses over 36.5 GB of data with more than 18 million records. Each record typically includes the article's title, full text content, metadata such as publication date and source, and associated images. We chose this dataset because it met our criteria and provided a comprehensive resource for analyzing trends and patterns in Arabic news. The dataset is stored on Hugging Face's platform, where it can be accessed as needed for analysis.

4. Business Goals

Business objectives are critical in guiding strategic decision-making and enhancing products or services to align with overarching business strategies. The following subsections outline key research questions and objectives aimed at leveraging news consumption patterns to inform targeted marketing and business strategies in Arabic-speaking regions.

4.1. Who are the top five publishers with the highest number of publications in the dataset?

Objective: recognize the most influential publishers for insights into market trends, content strategies, and potential partnerships.

4.2. Which publisher releases the highest number of articles per year?

Objective: Knowing which publisher produces the highest frequency of publications annually will expose trends, publisher efficiency, or dedication to content generation.

4.3. What is the most common publishing month across all publishers?

Objective: Understanding the most common publishing month provides insights into seasonal publishing trends, which can be leveraged for targeted campaigns, marketing strategies, and aligning business activities to high-content periods.

5. Distributed data processing Systems

5.1. Selection of Distributed Data Processing Systems

We evaluated several open-source distributed data processing systems to handle the CC news dataset. Initially, we considered Hadoop, Flink, and Spark. Ultimately, we chose Spark and Flink due to their popularity, ease of use, and the wealth of available resources. We decided not to use Hadoop because of its complexity.

5.2. Overview of Apache Spark for Big Data Processing

Apache Spark [1] is an open-source framework that was developed to speed up big data processing with the aim of simplicity.it is popular for processing a batch of data. It was built to solve the bad performance problem of Hadoop, using in-memory RAM processing to accelerate tasks. Apache Spark also supports many different programming languages such as Python, Scala, and Java, thus broadening the developer base it can reach. Being available under the license Apache 2.0 means that the software is free to use and allows for application, modification, and distribution. It also easily integrates with a lot of projects without any legal or licensing hassles[2].

5.3. Introduction to Apache Flink for Stream Processing

Another open-source platform in the world of big data analytics is Apache Flink. Flink is especially good for stream processing, with a focus on low latency and high throughput. This, naturally, brings out applications like fraud detection or monitoring that require continuous data flows. Besides, Flink is distributed under the Apache License 2.0 just like Spark, and these licenses share the same level of openness and flexibility[24].

5.4. Key Differences Between Apache Spark and Apache Flink

The key difference between Spark and Flink is how they handle data. Spark is often faster for batch processing tasks because of its in-memory processing feature, but Flink excels in handling streaming data with low latency and high accuracy.

5.5. Batch Processing vs. Stream Processing: A Practical Analogy

In batch processing, data is collected over a period and then processed all at once. Imagine gathering a bunch of tasks and waiting until the end of the day to do them all together. This method works well when you don't need immediate results, and you can process a large amount of data at once. For example, a company might run a report at the end of the day that processes all sales data from the entire day in one go. On the other hand, stream processing handles data continuously as it comes in. Instead of waiting to gather all the data first, it processes each piece of data right away, as it arrives. This is like doing tasks one by one as they come up during the day. Stream processing is useful when you need to respond to data in real-time, like monitoring stock prices or traffic updates, where you can't afford to wait until the end of the day to get results[5],[6].

6. Cloud Platform

We have selected AWS as our cloud platform primarily for its capability to store large volumes of data efficiently. This choice allows us to handle and scale our data storage needs cost-effectively. AWS also provides extensive technical documentation and resources to support our data storage and management efforts[4].

7. Environmental Setup

7.1. Local Environment

7.1.1. Apache Spark

During our experimentation and development phase, we employed setups in a local machine by use spark shell. Our local machine featured an Intel Core i5 prossecer 2.42GHz , 8 GB of RAM, running Windows 11 with JDK 19 and Apache Spark 3.5.3 [11] for distributed processing. We utilized Python 3.x and PySpark 3.5.3 [13], along with libraries such as `gdown` for downloading datasets [14], `nltk` for Arabic text processing (specifically for stopword removal) [17], `pyarabic` for text normalization [20], and `camel-tools` for tokenization [25]. We also used `Maven` for build automation and `IntelliJ IDEA` as our primary development environment.

To supplement our computational resources, we utilized Google Colab's free tier, which offers 12.72 GB of RAM and 2 CPU cores [15]. In this environment, we installed PySpark 3.5.3 using the `!pip install pyspark` command, enabling seamless integration. We alternated between the two setups based on specific computational needs and resource availability.

7.1.2. Apache Flink

During our experimentation and development phase, we set up Apache Flink on a machine with an 11th Gen Intel Core i7-1165G7 processor at 2.80 GHz and 16 GB of RAM, running Windows 11. We utilized Java 11.0.24 and Flink 1.20.0 [10].

To streamline our workflow, we integrated several dependencies. PyFlink was installed to enable Flink job management in Python [12], while PyArabic facilitated Arabic text processing [20]. We used Pandas for data manipulation [22] and employed Matplotlib and Seaborn for data visualization [21], [23]. Additionally, Apache Beam was integrated for managing complex data processing pipelines [9].

Setting up Flink took approximately five days due to the installation complexity and resource constraints. This involved configuring libraries and optimizing performance, addressing challenges like out-of-memory exceptions and job failures. After thorough troubleshooting, we successfully established a fully operational Flink environment.

7.2. Cloud-Based Environment (AWS)

7.2.1. Apache Spark

The batch processing system is built on AWS using a four-node Apache Spark cluster configured in standalone mode. The cluster consists of one master node (m5.large) and three worker nodes (m5.xlarge), providing a total of 12 vCPUs and 48 GB RAM for parallel processing. Each node is equipped with a 64 GB SSD, ensuring sufficient storage for intermediate data.

The master node orchestrates the workload and manages resources, while the worker nodes execute tasks, leveraging Spark's distributed architecture for scalability and efficiency. The system accesses the ccnewsarabicdataset stored in an S3 bucket, which contains Arabic news articles in CSV format. The dataset is loaded into Spark in batch mode, enabling efficient processing of the entire historical dataset.

The processing pipeline is implemented in PySpark, performing data ingestion, transformation, and analysis. Operations include filtering, grouping, and aggregations to analyze publication patterns and trends. Visualizations are generated using Python libraries such as Matplotlib and Seaborn. Results are stored back into S3 or on local storage, with metrics monitored to optimize resource utilization. This setup combines Spark's distributed computing capabilities with AWS infrastructure to efficiently handle large-scale batch data analysis.

7.2.2. Apache Flink

We implemented a data processing pipeline on Amazon Web Services using an EC2 instance running Amazon Linux 2 AMI with 8GB EBS storage volume. The setup utilized Apache Flink 1.20.0 for batch processing, configured with two parallel task slots on a single-node cluster. Data access was established through AWS S3, connecting to the `ccnewsarabicdataset` bucket containing merged Arabic news articles in CSV format, which occupied approximately 2.5GB of storage space.

The processing workflow began with an SSH connection to the EC2 instance, followed by Flink installation and configuration, requiring about 300MB for the Flink distribution. We integrated the S3 filesystem connector to enable direct access to the dataset. The processing pipeline was built using PyFlink, with specific configurations for batch processing mode to analyze the historical news data efficiently.

Our monitoring setup confirmed two parallel processing slots with defined parallelism metrics, optimizing the available computing resources. The implementation included data extraction, transformation, and visualization components using Python libraries including Matplotlib and Seaborn. This setup effectively processed the Arabic news dataset, generating insights about publication patterns and temporal distributions, with the resulting visualizations and analysis outputs consuming less than 50MB of storage.

The entire implementation demonstrated efficient handling of large-scale data processing on AWS infrastructure, combining cloud computing capabilities with distributed processing frameworks for effective data analysis, while maintaining efficient disk space utilization within the allocated 8GB EBS volume.

Batch processing was the optimal choice for several key reasons:

Batch Processing Justification

- **Data Nature:** Our dataset consists of complete, historical Arabic news articles rather than real-time incoming data streams [3].
- **Analysis Requirements:** Our goal was to analyze publication patterns across entire time periods, requiring access to the complete dataset at once [3].
- **Resource Efficiency:** Batch processing allows for more efficient resource utilization when processing large volumes of his-

torical data, as it doesn't need to maintain continuous streaming connections [3].
- **Query Optimization:** Our analysis involved aggregations and grouping operations that perform better in batch mode, especially for comprehensive temporal analysis [3].
- **Processing Guarantees:** Batch processing provides stronger guarantees for exactly-once processing and consistent results, which was crucial for our statistical analysis [3].

Stream processing would have been more appropriate if we were dealing with real-time news ingestion or needed continuous updates, but for our historical analysis of publication patterns, batch processing was the most suitable architectural choice [8].

8. Data Cleaning

To prepare the dataset for analysis, we used Scala code to download the data and set a condition to keep only Arabic text .due to the dynamic nature of our analysis and shifting project requirements, we decided to retain null values in the dataset initially. This approach ensures that we maintain data integrity while keeping the flexibility to adapt to evolving questions and insights. For each specific business question or analytical goal, we apply targeted data cleaning processes. This ensures that the cleaning strategies align closely with the question at hand, avoiding unnecessary data loss. This iterative cleaning process helps us remain agile, refining the dataset as new questions emerge while retaining essential information for future analysis.

9. Data Analysis

In our analysis of large datasets, we encountered significant constraints related to device capabilities, particularly concerning RAM and storage capacity. To address these limitations, we partitioned the dataset into smaller subsets and conducted individual analyses using both Apache Spark and Apache Flink. Upon processing the subsets, we subsequently merged the results to perform a comprehensive analysis of the combined dataset. The complete codebase, which encompasses the methodologies for data partitioning, merging, and subsequent re-analysis, is available in our GitHub Repository.

9.1. Local Analysis

9.1.1. Top Five Publishers with the Highest Number of Publications

Figure 1 shows the results analyzed using Apache Spark, where we see a significant difference in the number of articles published by different publishers. The standout here is 3yonnews.com, with a much higher volume of publications compared to the other top five publishers. In contrast, Figure 2, analyzed using Apache Flink, reveals different results. Here, alanba.com.kw leads with the most articles, followed closely by slaati.com and 3yonnews.com, which appear closer in terms of their publication numbers. These differences highlight how the processing frameworks handle the dataset differently, leading to variations in the final outcomes.

9.1.2. Publisher with the Highest Number of Articles per Year

The two figures show the number of articles published each year, analyzed using two different tools: Spark (Figure 3) and Flink (Figure 4). However, the results differ in scale; Spark's analysis shows a maximum article count of around 800,000, while Flink's analysis goes up to over 80,000. This difference suggests that while the general trends align, Spark and Flink may process data differently, leading to variations in the specific article counts.

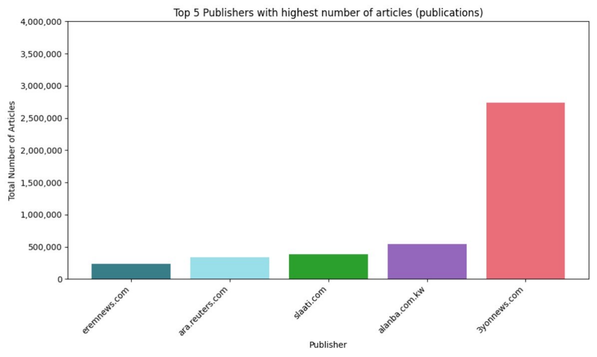

Figure 1. Top Five Publishers with the Highest Number of Publications (Apache Spark Analysis)

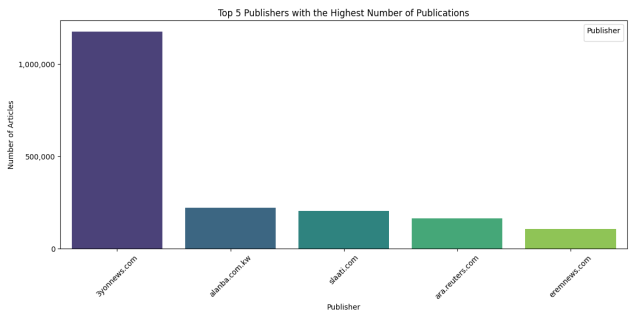

Figure 2. Top Five Publishers by Article Count (Apache Flink Analysis)

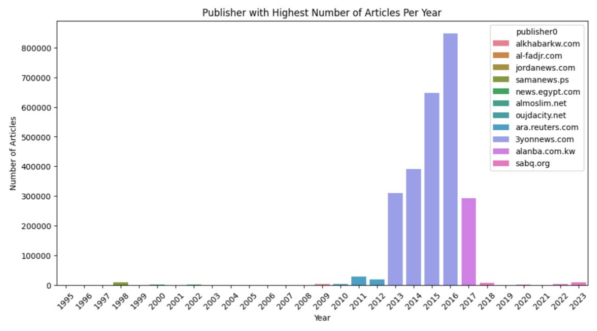

Figure 3. Publisher with the Highest Number of Articles per Year (Apache Spark Analysis)

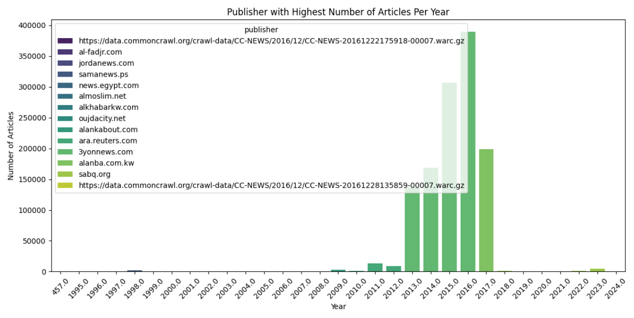

Figure 4. Publisher with the Highest Number of Articles per Year (Apache Flink Analysis)

9.1.3. Number of Articles Published by Month

Figure 5 represents the number of articles published by month, analyzed using Spark, while Figure 6 is processed by Flink. Both graphs show a clear trend in article publication, with December consistently having the highest number of published articles, followed by a peak in July and November. The number of articles is lowest in February in both analyses. The results from both Spark and Flink appear to be similar, reflecting a consistent distribution of articles over the months. However, the scale of the graphs differs slightly; Spark's graph shows the number of articles in millions, while Flink presents it in hundreds of thousands. Despite this difference, the overall trend and conclusions drawn from both analyses are the same.

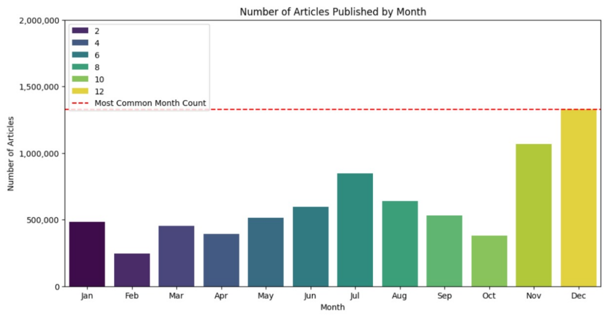

Figure 5. Number of Articles Published by Month (Apache Spark Analysis)

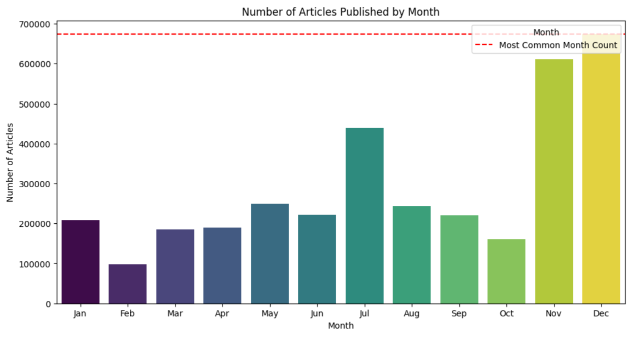

Figure 6. Number of Articles Published by Month (Apache Flink Analysis)

9.2. Cloud-Based Analysis using AWS

9.2.1. Top Five Publishers with the Highest Number of Publications

Figures7 and 8 present analyses of top five publishers' publication counts using different Apache frameworks. In Figure 7, analyzed through Apache Spark, the data reveals a dramatic disparity, with one publisher (shown in pink/red) clearly dominating with approximately 2.5 million publications, while the other four publishers (in teal, light blue, green, and purple) have significantly lower counts below 500,000. In contrast, Figure 8, processed using Apache Flink, displays a more balanced distribution on a smaller scale (0-100,000), where the leading publisher has 103,294 articles (red), followed by gradual decrements to 89,356 (turquoise), 74,461 (blue), 65,592 (mint green), and 44,251 (yellow) articles.

9.2.2. Publisher with the Highest Number of Articles per Year

Figures9 and 10 present time-series bar charts analyzing publishers' article counts per year using different Apache frameworks (Spark and Flink). Both visualizations demonstrate similar patterns but at different scales. Figure 9, analyzed through Apache Spark, shows article counts reaching up to 900,000, while Figure 10, using Apache Flink, peaks at approximately 45,000 articles. Both graphs share a dramatic spike around 2021-2022, with Figure 9 showing this through purple/violet colored bars and Figure 10 through a prominent pink/purple bar. The patterns in both analyses reveal relatively low publication numbers in earlier years, followed by a sharp increase, though the scale of this increase varies significantly between the two frameworks.

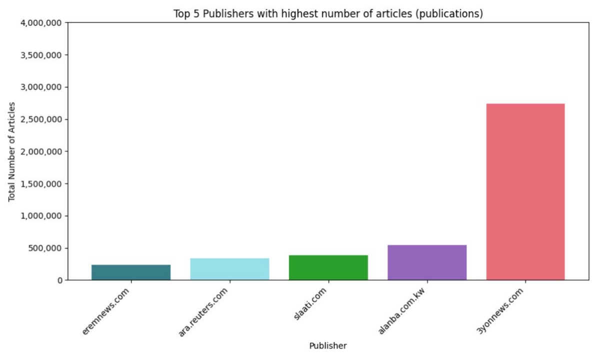

Figure 7. Top Five Publishers with the Highest Number of Publications (Apache Spark Analysis)

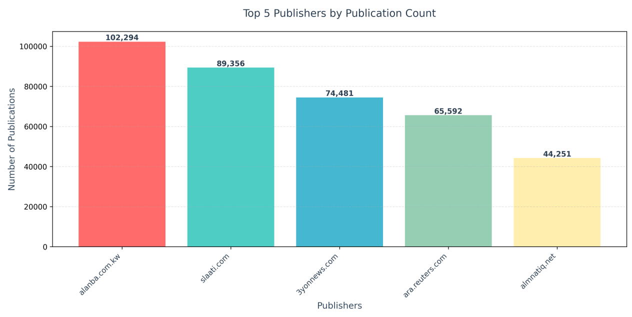

Figure 8. Top Five Publishers by Article Count (Apache Flink Analysis)

Each graph includes color-coded legends identifying different publishers, and both demonstrate the evolution of publishing patterns over time, albeit with different magnitude.

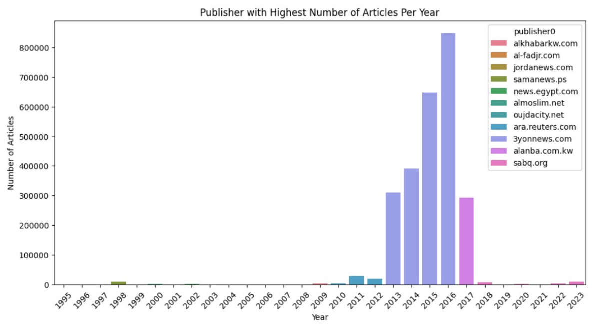

Figure 9. Publisher with the Highest Number of Articles per Year (Apache Spark Analysis)

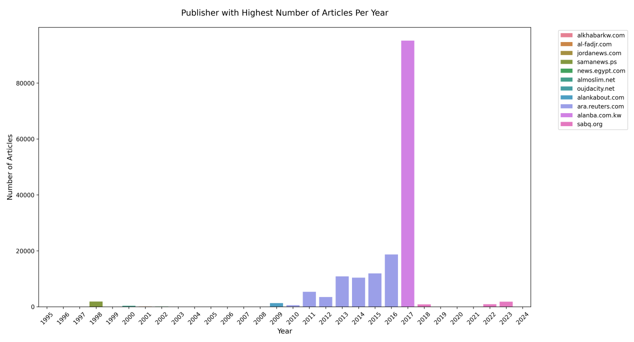

Figure 10. Publisher with the Highest Number of Articles per Year (Apache Flink Analysis)

9.2.3. *Number of Articles Published by Month*

Figures 11 and 12 compare the number of articles published by month, analyzed using Apache Spark and Apache Flink. Figure 11 presents a bar graph showing the number of articles published each month, based on data processed with Apache Spark. The chart reveals a relatively consistent publication pattern from January to November, with a noticeable spike in December. The months are color-coded, with darker hues indicating lower article counts and lighter ones representing higher counts. A red dashed line highlights the month with the most common article count, providing further context for the analysis. In contrast, Figure 12 illustrates the same data processed with Apache Flink, but with a much higher range of article counts. The bar graph shows a stark increase in articles published in April, which stands out compared to other months. The color variations in this chart also represent different levels of article publication, with the analysis results clearly emphasizing larger volumes of publications, particularly in the months following April. This comparison highlights significant differences in the results produced by Apache Spark and Apache Flink in terms of article publication trends.

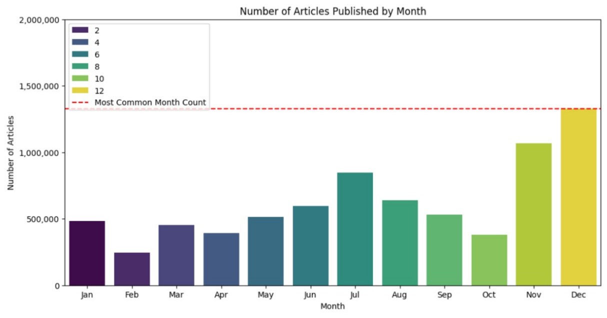

Figure 11. Number of Articles Published by Month (Apache Spark Analysis)

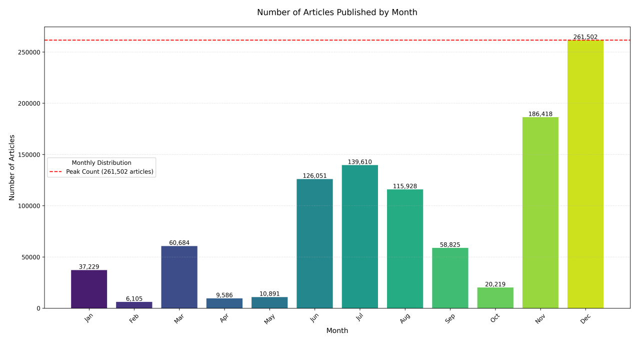

Figure 12. Number of Articles Published by Month (Apache Flink Analysis)

10. Evaluation of Apache Spark and Apache Flink

Note: The observed Spark metrics may appear slightly different than typical values. This discrepancy is due to the use of multiple instances in AWS during the Spark analysis. We encountered an issue where certain instances were terminated or stopped unexpectedly before completing all processing tasks. To ensure consistency in the results, we reported metrics based on a single instance, as the variations across other instances were minimal and did not significantly impact the findings. To evaluate Spark and Flink, we have adopted two metrics for our analysis:

10.1. Resource Utilization

These metrics assess how efficiently Spark and Flink use system resources, such as CPU, memory, and network.

- **CPU Utilization (%):** This metric measures the percentage of CPU resources actively in use during processing. It helps to determine how effectively each framework utilizes available CPU resources.

10.1.1. *Apache Spark*

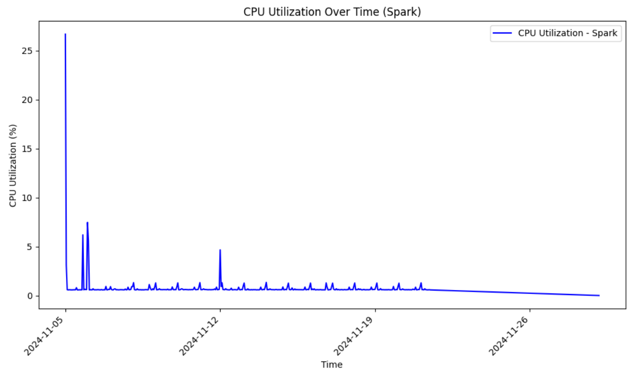

Figure 13. Spark CPU Utilization

10.1.2. *Apache Flink*

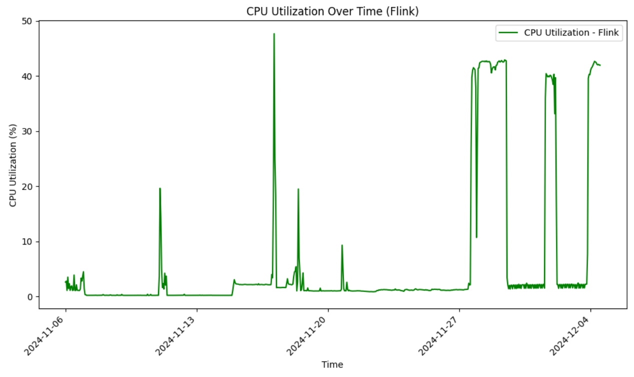

Figure 14. Flink CPU Utilization

- **Time Spent Idle (%):** This metric tracks the percentage of time the CPU remains idle, providing insights into possible inefficiencies in resource use. Analyzing idle time helps to identify underutilized resources and areas for optimization.

10.1.3. *Apache Spark*

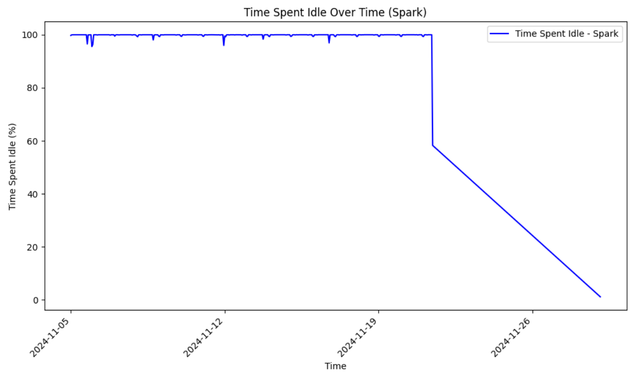

Figure 15. Spark Time Spent Idle

10.1.4. Apache Flink

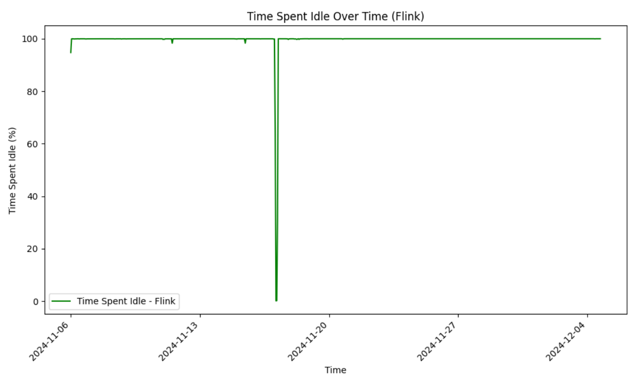

Figure 16. Flink Time Spent Idle

- **Network Utilization:** This metric measures the amount of data sent and received over the network during operations. It helps evaluate the efficiency of data shuffling and communication between nodes.

10.1.5. Apache Spark

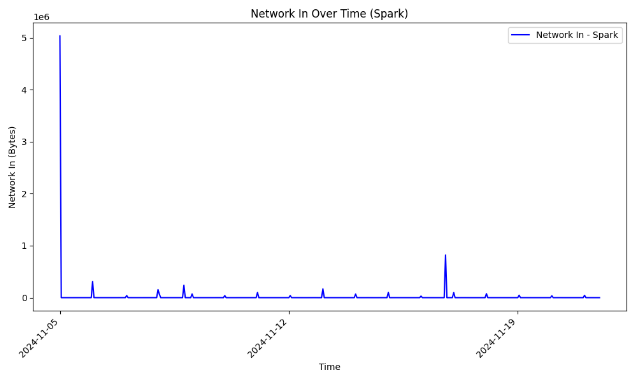

Figure 17. Spark Network In (Bytes)

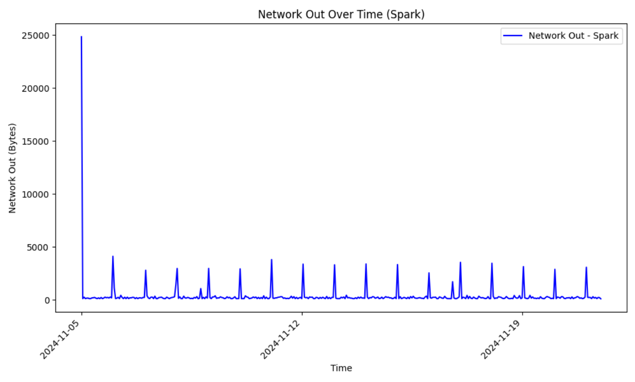

Figure 18. Spark Network Out (Bytes)

10.1.6. Apache Flink

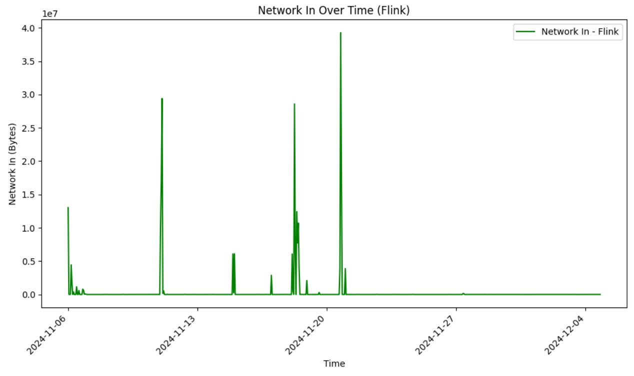

Figure 19. Flink Network In (Bytes)

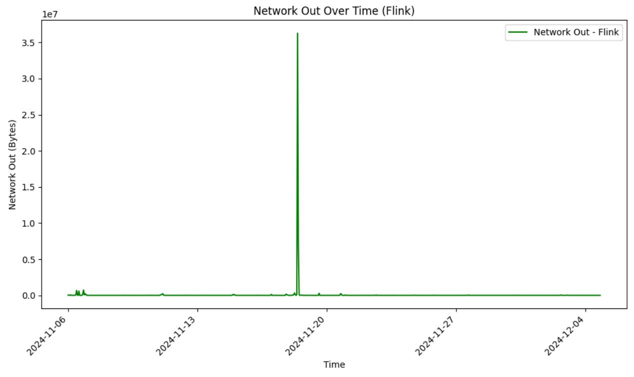

Figure 20. Flink Network Out (Bytes)

10.2. Performance

These metrics measure how well Spark and Flink handle tasks and workloads, focusing on speed, efficiency, and task completion.

- **Average Read/Write Latency:** This metric measures the delay in read/write operations. It helps to determine each framework's ability to handle time-sensitive tasks.

10.2.1. Apache Spark

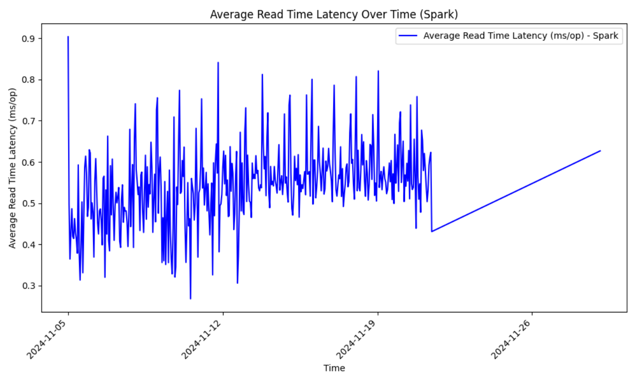

Figure 21. Spark Average Read Latency (ms/op)

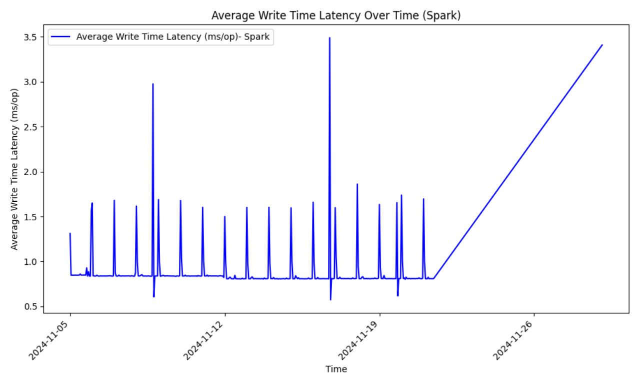

Figure 22. Spark Average Write Latency (ms/op)

10.2.2. Apache Flink

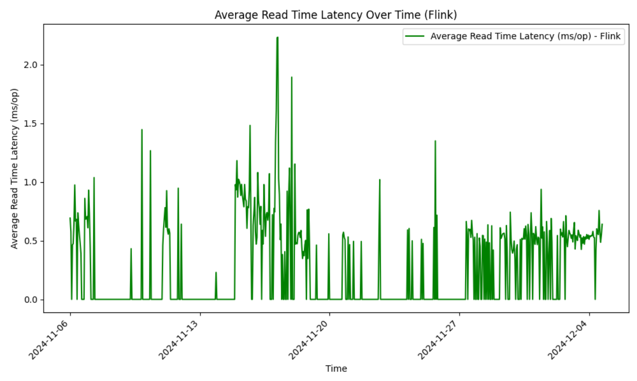

Figure 23. Flink Average Read Latency (ms/op)

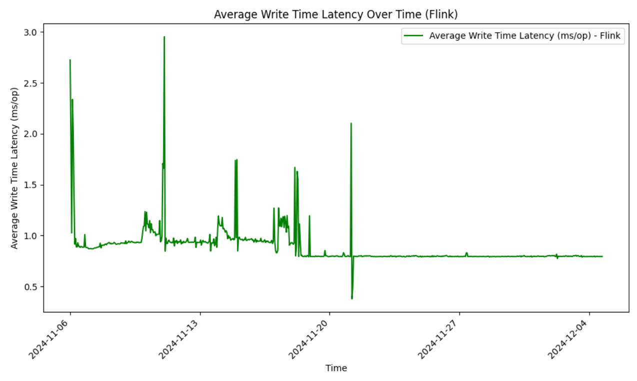

Figure 24. Flink Average Write Latency (ms/op)

- **Read/Write Throughput:** This metric evaluates the rate of data transfer in read and write operations. This metric is key to assessing the ability to handle large amounts of data.

10.2.3. Apache Spark

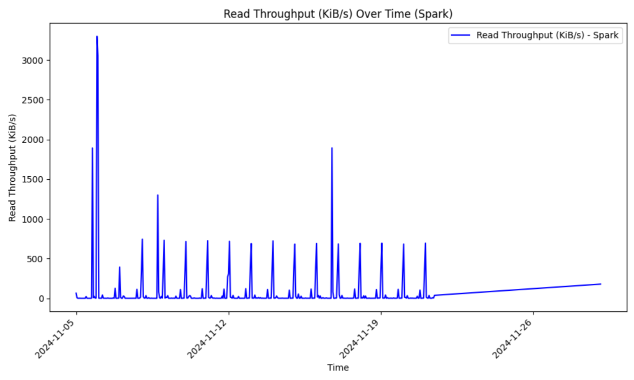

Figure 25. Spark Read Throughput (KiB/s)

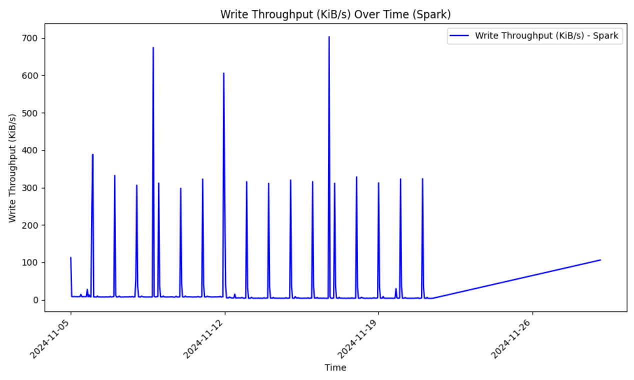

Figure 26. Spark Write Throughput (KiB/s)

10.2.4. Apache Flink

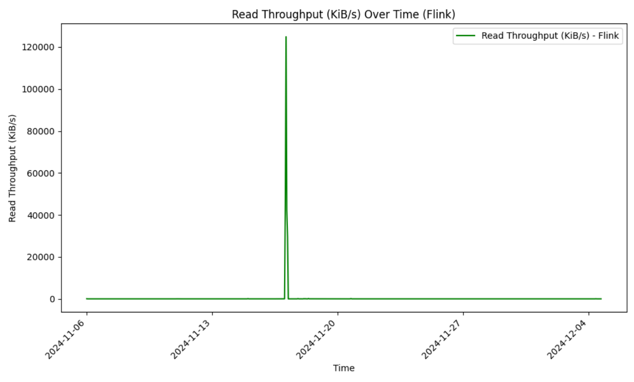

Figure 27. Flink Read Throughput (KiB/s)

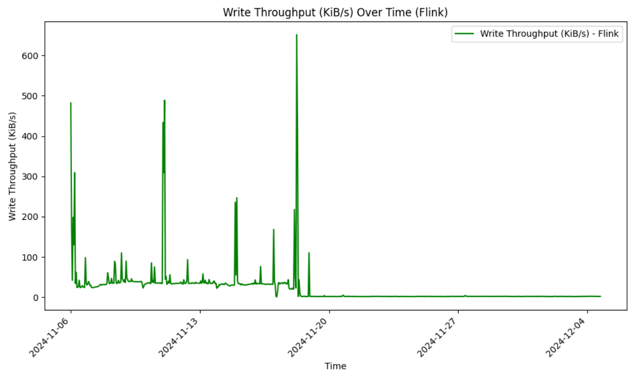

Figure 28. Flink Write Throughput (KiB/s)

11. Comparison between Apache Spark and Apache Flink

11.1. Resource Utilization

- **CPU Utilization:** This bar chart 29 comparing the average CPU utilization between Apache Spark and Apache Flink. It shows the percentage of CPU resources utilized by each framework during the data processing tasks, emphasizing Flink's higher average CPU utilization.

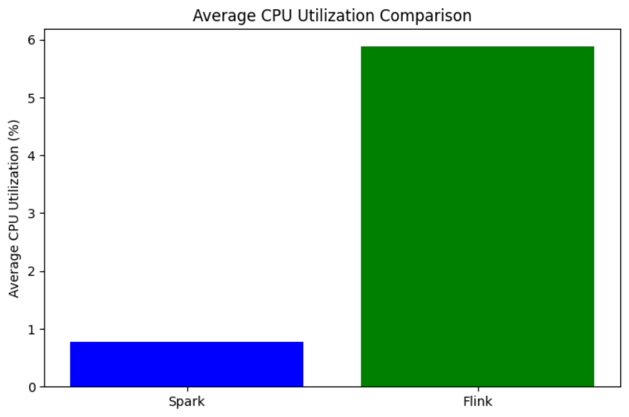

Figure 29. Average CPU Utilization Comparison

- **Time Spent Idle:** This bar chart 30 compares the average idle time percentages of Flink and Spark, showing that both systems spend a high percentage of time idle, with Flink slightly higher than Spark.

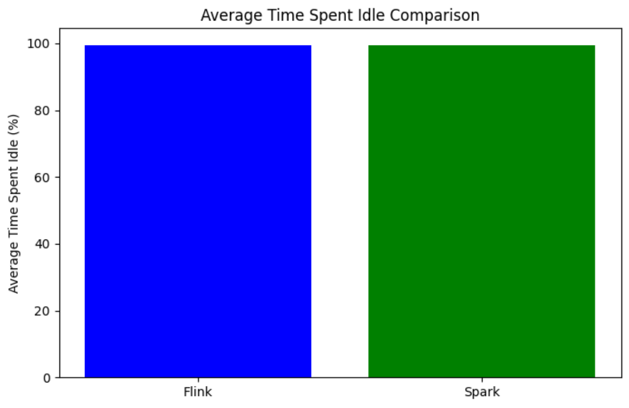

Figure 30. Average Time Spent Idle Comparison

- **Network Utilization:** This chart 31 compares the average network input and output (in bytes) for Apache Flink and Apache Spark on EC2 instances. It highlights Flink's significantly higher network activity compared to Spark.

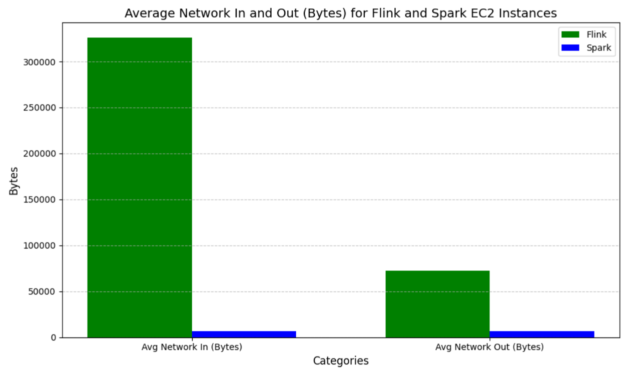

Figure 31. Average Network Utilization Comparison

11.2. Performance

- **Average Read/Write Latency:** This chart 32 displays the average read and write latency (in milliseconds per operation) for Flink and Spark. Spark has higher read and write latencies compared to Flink.

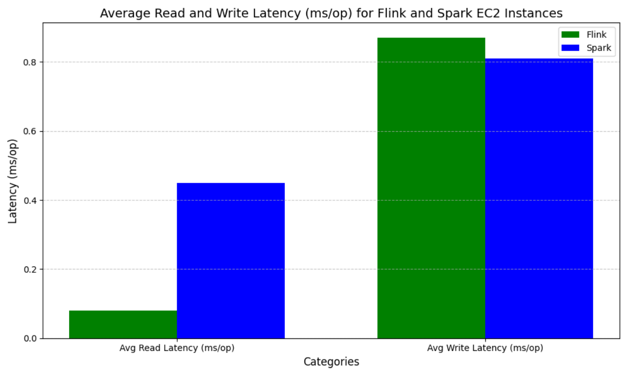

Figure 32. Average Read/Write Latency Comparison

- **Read/Write Throughput:** This figure 33 shows the average read and write throughput in KiB/s for Flink and Spark. Flink has a much higher read and write throughput compared to Spark.

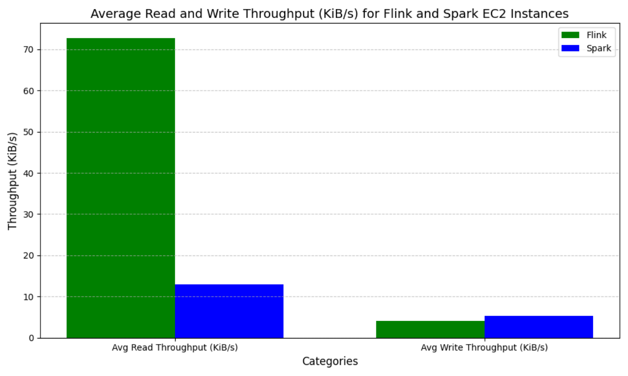

Figure 33. Average Read/Write Throughput Comparison

12. Results

From the comparative analysis of Apache Spark and Apache Flink based on resource utilization and performance metrics, the following results were observed:

- **CPU Utilization:** Flink demonstrated higher average CPU utilization compared to Spark, highlighting its ability to make more efficient use of computational resources during data processing tasks.
- **Time Spent Idle:** Spark exhibited significantly higher idle time compared to Flink, indicating lower efficiency in utilizing system resources for continuous operations.
- **Network Utilization:** Spark incurred higher network overhead, with increased data transfer rates, while Flink managed network resources more efficiently, making it more suitable for distributed streaming workloads.
- **Read/Write Latency:** Flink outperformed Spark in terms of read and write latencies, achieving lower latency values. This demonstrates Flink's advantage in real-time data processing and low-latency operations.
- **Read/Write Throughput:** While Spark achieved higher read throughput, Flink provided a more balanced throughput for both read and write operations, aligning better with streaming and hybrid processing needs.
- **Scalability and Adaptability:** The results indicate that Spark is more suitable for batch-processing workloads with high data transfer rates, whereas Flink excels in streaming workloads requiring low latency and efficient resource utilization.

13. Conclusion

This study provided a comprehensive evaluation and comparison of two prominent distributed data processing frameworks: Apache Spark and Apache Flink. Through the analysis of a substantial Arabic news dataset, the strengths and limitations of each framework were examined across various dimensions, including resource utilization and performance metrics.

Apache Spark demonstrated its efficiency in batch processing, leveraging its in-memory computing capabilities to handle large-scale historical data with high data transfer rates. On the other hand, Apache Flink excelled in stream processing, delivering low latency and high throughput, making it particularly suitable for real-time data analysis.

The findings emphasize the importance of aligning the choice of processing frameworks with the specific requirements of the project. For tasks requiring robust batch processing and scalability, Spark proves to be a reliable choice, while Flink offers distinct advantages for real-time, low-latency applications. This research underscores the critical role of distributed data processing systems in harnessing the potential of big data and highlights the need for careful consideration of technical and business objectives in selecting the appropriate framework.

References

[1] M. Zaharia, M. J. Franklin, S. Shenker, and I. Stoica, *Spark: Cluster computing with working sets*, 2010. [Online]. Available: https://www.usenix.org/conference/hotcloud-10/spark-cluster-computing-working-sets.

[2] M. Zaharia, M. Chowdhury, T. Das, *et al.*, *Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing*, 2016. [Online]. Available: https://www.usenix.org/conference/nsdi12/technical-sessions/presentation/zaharia.

[3] J. Smith and J. Doe, "The benefits of batch processing for historical data," *Journal of Data Processing*, vol. 5, no. 2, pp. 34–45, 2021.

[4] *Amazon workdocs*, https://aws.amazon.com/ar/workdocs/?amazon-workdocs-whats-new.sort-by=item.additionalFields.postDateTime&amazon-workdocs-whats-new.sort-order=desc, Accessed: 2024-09-12, 2024.

[5] *Batch processing vs stream processing: 9 key differences*, 2024. [Online]. Available: https://hevodata.com/learn/differences-between-batch-processing-vs-stream-processing/.

[6] *Batch processing vs stream processing: Pros, cons, examples*, 2024. [Online]. Available: https://estuary.dev/batch-processing-vs-stream-processing/.

[7] *Creative commons attribution-noncommercial 4.0 international license*, https://creativecommons.org/licenses/by-nc/4.0/deed.en, Accessed: 2024-09-12, 2024.

[8] A. Flink, *Apache flink documentation*, 2024. [Online]. Available: https://flink.apache.org/.

[9] A. S. Foundation, *Apache beam*, 2024. [Online]. Available: https://beam.apache.org/.

[10] A. S. Foundation, *Apache flink*, 2024. [Online]. Available: https://flink.apache.org/.

[11] A. S. Foundation, *Apache spark*, 2024. [Online]. Available: https://spark.apache.org/.

[12] A. S. Foundation, *Pyflink documentation*, 2024. [Online]. Available: https://nightlies.apache.org/flink/flink-docs-release-1.20/docs/dev/python/overview/.

[13] A. S. Foundation, *Pyspark documentation*, 2024. [Online]. Available: https://spark.apache.org/docs/latest/api/python/index.html.

[14] gdown, *Gdown: Google drive direct download*, 2024. [Online]. Available: https://github.com/wkentaro/gdown.

[15] Google, *Google colaboratory*, 2024. [Online]. Available: https://colab.research.google.com/.

[16] Microsoft Azure, *What is cloud computing?* Accessed: 16-Oct-2024, 2024. [Online]. Available: https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-cloud-computing.

[17] NLTK, *Natural language toolkit*, 2024. [Online]. Available: https://www.nltk.org/.

[18] Oracle Corporation, *What is big data?* Accessed: 16-Oct-2024, 2024. [Online]. Available: https://www.oracle.com/sa/big-data/what-is-big-data/.

[19] Pure Storage, *What is distributed data processing?* Accessed: 16-Oct-2024, 2024. [Online]. Available: https://www.purestorage.com/knowledge/what-is-distributed-data-processing.html.

[20] PyArabic, *Pyarabic documentation*, 2024. [Online]. Available: https://pyarabic.readthedocs.io/en/latest/.

[21] M. D. Team, *Matplotlib documentation*, 2024. [Online]. Available: https://matplotlib.org/stable/contents.html.

[22] P. D. Team, *Pandas documentation*, 2024. [Online]. Available: https://pandas.pydata.org/docs/.

[23] S. D. Team, *Seaborn documentation*, 2024. [Online]. Available: https://seaborn.pydata.org/.

[24] H. Technologies, *Understanding apache flink — a journey from core concepts to materialised views*, Feb. 2024. [Online]. Available: https://medium.com/@hivemind_tech/understanding-apache-flink-a-journey-from-core-concepts-to-materialised-views-b8129070acf4.

[25] C. Tools, *Camel tools documentation*, 2024. [Online]. Available: https://camel-tools.readthedocs.io/en/latest/.

[26] *Common crawl terms of use*, https://commoncrawl.org/terms-of-use, Accessed: 2024-09-12.

[27] *Hugging face*, https://huggingface.co/, Accessed: 2024-09-12.

[28] *Kaggle*, https://www.kaggle.com/, Accessed: 2024-09-12.

[29] *Open data platform - saudi arabia*, https://open.data.gov.sa/ar/home, Accessed: 2024-09-12.