Dask vs. Spark: Performance Analysis in Local and Cloud Environments.

Munirah Alobaid 443201024
Norah Alarifi 443200443
Danah Aljurayyan 443200585
Lujain Fatani 443203116
Ruba Alhuwaidi 443200952

December 5, 2024

1 Introduction

Distributed data processing systems have become essential for handling large-scale data, offering various degrees of scalability, performance, and efficiency. With the surge in data volume, velocity, and variety, advanced frameworks are crucial for deriving insights from massive datasets.
This research evaluates and compares two key distributed data processing systems: Dask and Apache Spark. We will assess their performance in both local and cloud environments (AWS) [1] focusing on:

- Performance: Assessing the speed and efficiency of data processing with a fixed dataset.
- Fault Tolerance: Evaluating how each system handles and recovers from failures.
- Ease of Use: Comparing the learning curve and developer experience for each system.
- Cloud Integration: Analyzing how well each system utilizes cloud resources.

2 Dataset

The first step in evaluating these two distributed data processing systems is choosing a dataset to test. It was challenging to satisfy the criteria for having a large dataset of over 10 million records, that includes Arabic text, images, and timestamps.

Our group members started to search for the dataset in multiple open-source resources, we looked for valuable datasets that would generate good business questions later.

As we searched, we found in Hugging Face a larger dataset of standard canvas [2] of over 10 million records (17 TB in size), images included. However, we excluded it for several reasons: 17 TB is too large to handle, especially with our limited resources, and we wanted a dataset from which we could extract business goals.

Unfortunately, finding a dataset that satisfies these requirements was very hard and almost impossible, which led us to think about merging multiple datasets or using an API. The open data platform[3] contained multiple small Arabic, timestamped datasets and it was very difficult to merge because of their small sizes. So, we filled out the Open Data Request form, asking for any dataset that meets our criteria regardless of the information behind the data. SDAIA got back to us, saying that they don't have a dataset with this size. We thought about merging Arabic Tweets [4] datasets from Kaggle [5] despite the diversity of its topics such as Egyptian [6], financial, and political tweets [7], but still, the resulting dataset wouldn't exceed 2 million records.

Although using API for tweets or anything else offered several advantages. However, complex querying and API rate limits were a real issue and due to our time limitations, traditional methods like downloading files from dataset resources may be more efficient.

Finally, we decided to choose a dataset for eCommerce behavior data from multi category store [8] from Kaggle[5], even though it didn't follow the Arabic text and image conditions, it gave us valuable data that answers our business goals to process.

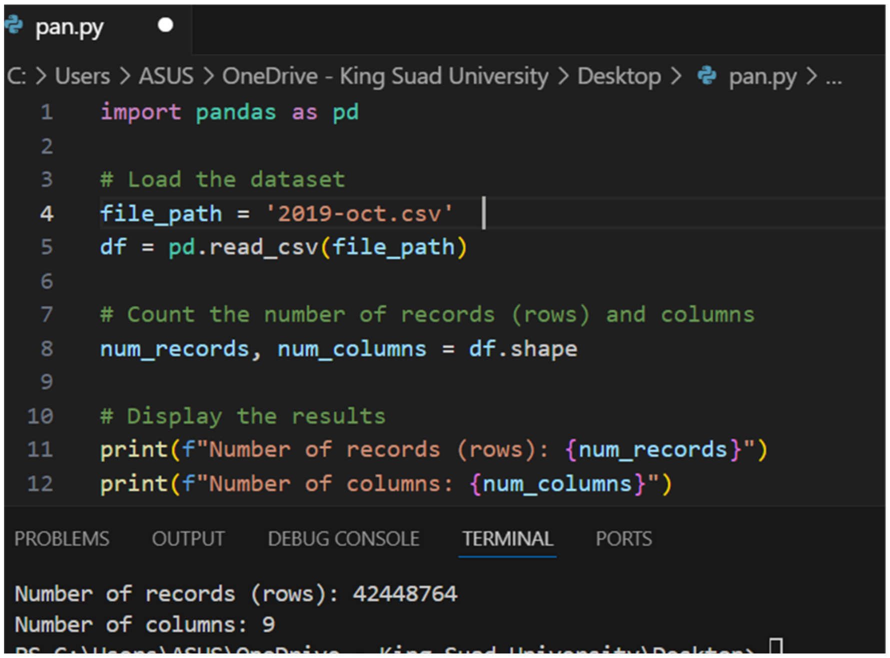
Our chosen dataset was found in Kaggle [5]. Contains eCommerce behavior data from multi-category stores for seven months (from October 2019 to April 2020) [9], collected by the Open CDP [10] on a large eCommerce store (15M visitors per month), we chose to focus on the October 2019 file to ensure our analysis remains manageable within the computational resources available on our local machines. The October 2019 file still provides substantial data (more than 42 million records) for meaningful analysis. All sensitive and personal data was removed. Broken data (like NULL price or products without categories) was removed, too. Thanks to the REES46 Marketing Platform [11] for this dataset and for allowing us to use it for educational purposes.

Dataset Information

- Dataset size:
  The size of the dataset (the October 2019 file only) is 5.3 GB.

  Number of records\rows: 42,448,764 .

  Number of columns: 9.

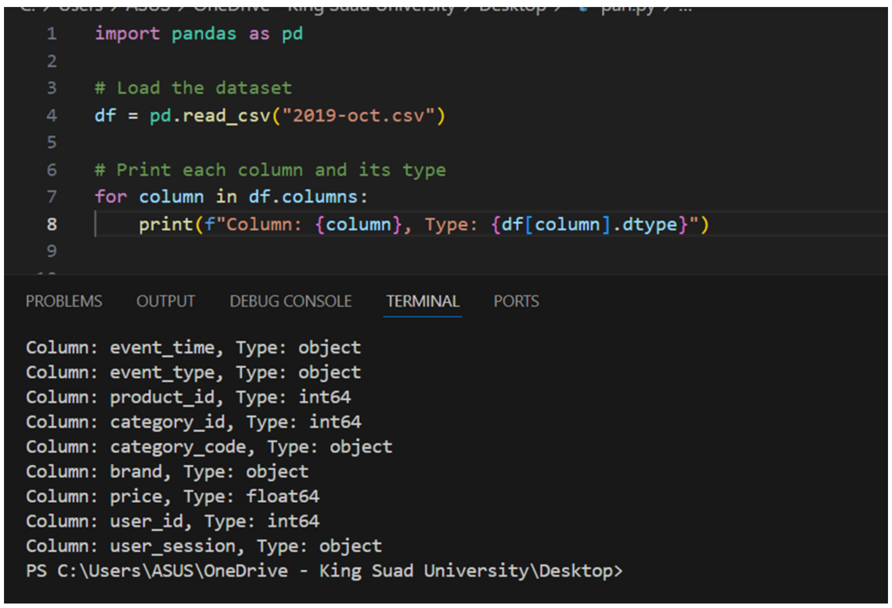

pan.py
C: > Users > ASUS > OneDrive - King Suad University > Desktop > pan.py > ...
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS

```python
import pandas as pd

# Load the dataset
file_path = '2019-oct.csv'
df = pd.read_csv(file_path)

# Count the number of records (rows) and columns
num_records, num_columns = df.shape

# Display the results
print(f"Number of records (rows): {num_records}")
print(f"Number of columns: {num_columns}")
```

```
Number of records (rows): 42448764
Number of columns: 9
```

Figure 1: A screenshot shows the counting of the number of records and columns of the dataset

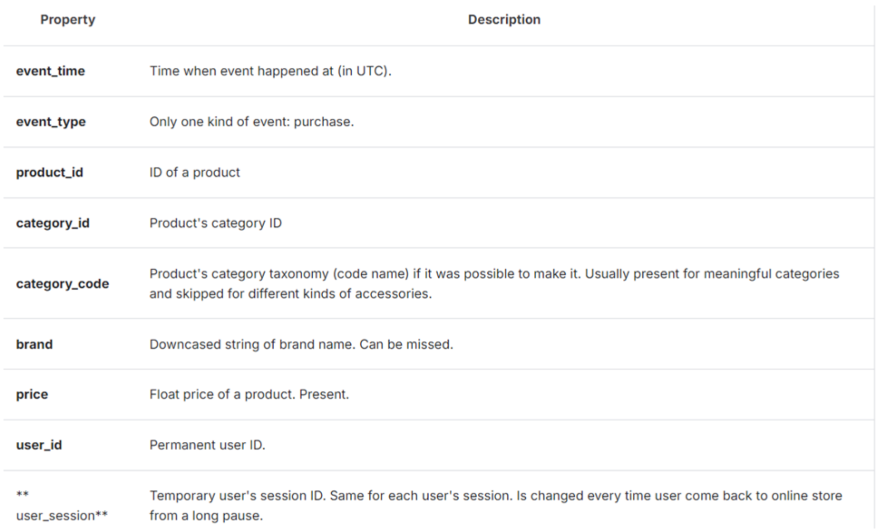
- Dataset language: English
- Data type and description: The dataset consists of the text of event time and type, product ID, category ID, category code, brand, price, user ID, and user session, shown in figures 2, and 3 each column and its type with a brief description of each attribute.



```python
import pandas as pd

# Load the dataset
df = pd.read_csv("2019-oct.csv")

# Print each column and its type
for column in df.columns:
    print(f"Column: {column}, Type: {df[column].dtype}")

```

```
Column: event_time, Type: object
Column: event_type, Type: object
Column: product_id, Type: int64
Column: category_id, Type: int64
Column: category_code, Type: object
Column: brand, Type: object
Column: price, Type: float64
Column: user_id, Type: int64
Column: user_session, Type: object
PS C:\Users\ASUS\OneDrive - King Suad University\Desktop>
```

Figure 2: A screenshot shows each column in the dataset and its type.



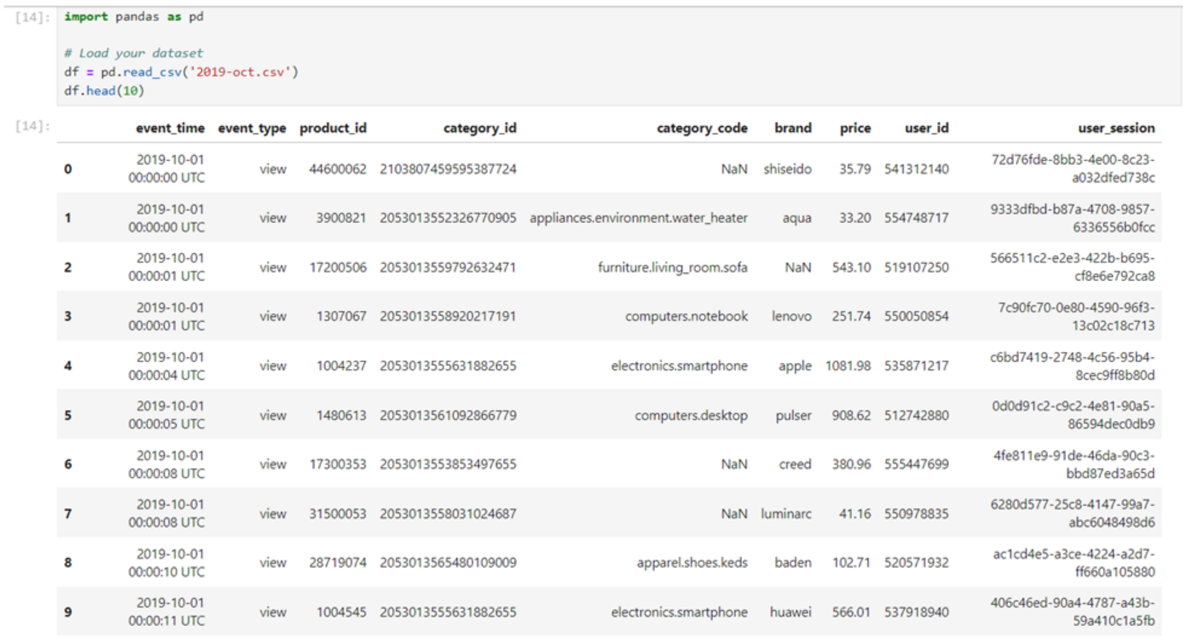
Property	Description
event_time	Time when event happened at (in UTC).
event_type	Only one kind of event: purchase.
product_id	ID of a product
category_id	Product's category ID
category_code	Product's category taxonomy (code name) if it was possible to make it. Usually present for meaningful categories and skipped for different kinds of accessories.
brand	Downcased string of brand name. Can be missed.
price	Float price of a product. Present.
user_id	Permanent user ID.
** user_session**	Temporary user's session ID. Same for each user's session. Is changed every time user come back to online store from a long pause.

Figure 3: Screenshot showing the description of each column or attribute

Note that the event's type can be:
- view (when a user views a product)
- cart (when a user adds a product to his\her cart)
- remove_from_cart (when a user removes a product out of his\her cart)
- purchase (when a user purchases a product)

- Data sample:

We used Jupyter Notebook [12] to print a sample of our dataset using a Python code (only the first 10 records), shown in Figure 4.



```
[14]: import pandas as pd

      # Load your dataset
      df = pd.read_csv('2019-oct.csv')
      df.head(10)
```

[14]:

	event_time	event_type	product_id	category_id	category_code	brand	price	user_id	user_session
0	2019-10-01 00:00:00 UTC	view	44600062	2103807459595387724	NaN	shiseido	35.79	541312140	72d76fde-8bb3-4e00-8c23-a032dfed738c
1	2019-10-01 00:00:00 UTC	view	3900821	2053013552326770905	appliances.environment.water_heater	aqua	33.20	554748717	9333dfbd-b87a-4708-9857-6336556b0fcc
2	2019-10-01 00:00:01 UTC	view	17200506	2053013559792632471	furniture.living_room.sofa	NaN	543.10	519107250	566511c2-e2e3-422b-b695-cf8e6e792ca8
3	2019-10-01 00:00:01 UTC	view	1307067	2053013558920217191	computers.notebook	lenovo	251.74	550050854	7c90fc70-0e80-4590-96f3-13c02c18c713
4	2019-10-01 00:00:04 UTC	view	1004237	2053013555631882655	electronics.smartphone	apple	1081.98	535871217	c6bd7419-2748-4c56-95b4-8cec9ff8b80d
5	2019-10-01 00:00:05 UTC	view	1480613	2053013561092866779	computers.desktop	pulser	908.62	512742880	0d0d91c2-c9c2-4e81-90a5-86594dec0db9
6	2019-10-01 00:00:08 UTC	view	17300353	2053013553853497655	NaN	creed	380.96	555447699	4fe811e9-91de-46da-90c3-bbd87ed3a65d
7	2019-10-01 00:00:08 UTC	view	31500053	2053013558031024687	NaN	luminarc	41.16	550978835	6280d577-25c8-4147-99a7-abc6048498d6
8	2019-10-01 00:00:10 UTC	view	28719074	2053013565480109009	apparel.shoes.keds	baden	102.71	520571932	ac1cd4e5-a3ce-4224-a2d7-ff660a105880
9	2019-10-01 00:00:11 UTC	view	1004545	2053013555631882655	electronics.smartphone	huawei	566.01	537918940	406c46ed-90a4-4787-a43b-59a410c1a5fb

Figure 4: Screenshot showing a dataset sample using a script in Jupyter Notebook.

- Data License: We used data files provided by the original authors [13].
- Expected update frequency: Never (Updated 5 years ago)
- Dataset storing:
  Our dataset is stored in some of our machines , we also uploaded it as an object in AWS S3's bucket [14] , and as GCS bucket [15].

3 Business Goals

The primary objective of this project is to compare the performance of the two data processing systems—Spark and Dask—using an e-commerce dataset. By performing data analytics on the dataset, we want to assess and compare those two systems' handling of large-scale data processing tasks to determine which system may be more suitable for similar use cases. [16].

3.1 Question 1:

Which product categories have the highest conversion rates?

Goal: Optimize Product and Category Focus.

Objective: Identify product categories that perform the best in terms of turning views into purchases; this allows businesses to focus most of their resources on high-performing categories

3.2 Question 2:

How do product price ranges within each category affect customer engagement (product viewing or adding items to the cart) and conversion rates?

Goal: Enhance Pricing Strategies.

Objective: Analyse how product price ranges influence customer interactions and conversions, allowing businesses to create an optimal pricing model.

4 Distributed Data Processing Systems

Extracting meaningful information from the dataset, also known as data analytics, is crucial. Therefore, choosing the most appropriate distributed data processing system is critical to ensure fault tolerance, performance, scalability, and resource utilization since we will mostly evaluate the two systems based on these factors. When handling large datasets, the system can impact both the speed and quality of insights derived from the data.

Apache Spark [17] and Hadoop [18] were the first options that have been considered due to their prominence, yet after some research, we found out that there are newer options that are worth considering like Dask[19] and clickHouse[20].

ClickHouse ClickHouse is an open-source, columnar database management system that was introduced by Yandex in 2016 [21]. It is known for its exceptional performance in processing analytical queries on large datasets [20]. However, we opted not to use ClickHouse in the project for two primary reasons. Firstly, its relatively new presence in the market means that the community around it is still growing, resulting in fewer resources, plugins, and integrations compared to more mature database systems like MySQL or PostgreSQL [22]. A smaller community can hinder the ability to troubleshoot issues or find expert support during development.

The Apache Hadoop software library enables distributed processing of large datasets across clusters of computers using straightforward programming models. It scales from single servers to thousands of machines, providing both computation and storage. Hadoop ensures high availability by handling failures at the application level, rather than relying on hardware, making it resilient to individual machine failures [18].

In terms of processing speed, Hadoop has become outdated compared to newer technologies like Apache Spark. Since Hadoop used to depend on batch processing, the response time for analytics is low. We might need to use platforms outside of Hadoop to view data in real-time [23].

In 2012 Hadoop 2.0 was presented which broadly consists of two components **Hadoop Distributed File System(HDFS)** which can be used to store large volumes of data and **Yet Another Resource Negotiator(YARN)** which provides resource management and scheduling for running jobs.

Although it is very fault tolerant, it does not tend to be the best option out there due to its slower performance in processing large datasets compared to alternatives like Apache Spark. Therefore, other options are considered. It allows you to use, modify, and distribute the licensed software, including creating derivative works, without requiring those derivative works to be licensed under the same terms.

Apache Spark is an open-source engine built for large-scale data analysis. It can run on both single-node systems and extensive clusters, functioning as a multi-language platform for data engineering, data science, and machine learning. Its built-in support for in-memory distributed processing and fault tolerance enables users to create intricate, multi-stage data pipelines with ease and efficiency. It has in-memory computing features that provides quick processing, a generalized execution model to support various applications. It also has Java, Scala, Python, and R APIs [24].

Apache Spark stands out from other open-source platforms due to its integrated web user interface, known as Spark UI. This tool allows experts to monitor their Spark clusters' status and resource consumption. It is widely used in various industries, including gaming, streaming services, healthcare, e-commerce, etc. It is used by firms like **Alibaba, eBay, TripAdvisor and Pinterest.**

Apache Spark, Apache Hadoop, and ClickHouse are all distributed under the Apache License 2.0. This open-source license permits users to freely use, modify, and distribute the software for both commercial and non-commercial purposes, provided certain conditions are met. The full license text for Apache Spark can be accessed here, for Apache Hadoop here, and for ClickHouse here.

Dask is an open-source parallel computing library. It started as a tool to parallelize NumPy [25] operations on a single machine but has evolved into a powerful library for handling big data. It efficiently manages large arrays, Pandas DataFrames, and machine learning tasks, overcoming memory and computation limits. With its distributed scheduler, Dask now handles massive, multi-terabyte workloads across multiple machines [26].

It is written using Python which makes handling large amounts of data easier, Dask also uses batching techniques which makes the job much easier and gives us one-click access to our data**. D**ask can handle multiple types of data in a single stream [27].

Dask is also widely used by big firms like NASA, Walmart, and Microsoft. Although it is not the most widely used yet it is the third tool after spark and Hadoop [19].

Dask is licensed under the **BSD 3-Clause License** , which is a permissive open-source license that allows free use, modification, and distribution of software. It requires minimal restrictions, such as retaining copyright notices and disclaimers, and does not impose obligations on derivative works. It is business-friendly and compatible with other licenses, making it flexible for both open-source and commercial use, while also offering no warranties for the software.

Ultimately, we decided to use Dask for its usability and since it is very compatible with Python, the second tool is Spark for various reasons one of which is the fact that it is widely used, unlike Hadoop which lost its spark next to Apache Spark.

5 Cloud platform

We initially decided to use Amazon Web Services [1], as it delivers reliable and scalable services such as S3 [14], EC2 [28], and Redshift [29]. but we encountered unforeseen issues that affected our flow, which forced us to rethink our choices. A primary challenge we faced was the complexity of AWS's identity and access management (IAM) [30] system. IAM is a powerful tool for assigning permissions, but it requires careful role-based configurations for each service, resulting in a time-consuming setup. For example, during our set-up, granting necessary permissions to access S3 [14] required specific rules and policies that were quite difficult to manage effectively. The strict default settings were intended to enhance security, but they made it more challenging to configure appropriate access permissions, particularly when public access or external integrations were required. which is a trade-off, understandably, between security and ease of use, but it wasn't a good fit for us. Credential management also presented some difficulties. AWS requires multiple access keys and careful handling of credentials. As our set-up scaled across multiple nodes, it became pretty difficult to manage and distribute these access keys securely. Finally, AWS's billing and cost management system [31] proved a problem as we encountered unexpected costs, which required constant monitoring of active services to avoid incurring extra costs. This issue added a burden that detracted us from our primary focus.
Ultimately, these challenges prompted us to explore alternative solutions, one of which was changing our cloud platform to Google Cloud Platform (GCP)[32]. By creating, and running virtual machine instances, and grouping them in clusters to perform our analysis in an efficient way we anticipated a more streamlined experience, avoiding the complexities we had initially encountered with AWS[1].

6 Local Environment Setup

This subsection details setting up Apache Spark and Dask on our local machine for the initial data analysis. We ensured compatibility and optimal performance by carefully following installation procedures and configuring our environment appropriately.

Hardware Specifications:

- Operating System: Ubuntu 24.04.1 LTS
- Operating System type: 64-bit
- Processor: 4 cores, and 8-thread-support
- Memory: 16.0 GiB

6.1 Apache Spark Setup

- **Java:** We installed OpenJDK version 17.0.12 [33], an open-source Java implementation required to run Spark since it operates on the Java Virtual Machine (JVM).
- **Scala:** Spark is written in Scala [34], and its native integration with Spark APIs and ecosystems makes it feel like the most natural way to interact with Spark. Our setup used Spark version 3.2.0, downloaded from the official Apache Spark website [35], which comes bundled with Scala 2.12.18.
- **Spark Shell:** We set up the environment using Spark Shell in our terminal to interact with Spark directly from the command line. This allowed us to test Spark operations in a simplified interactive session.
- **Visualization of the results:** The results of Spark's analysis were saved into a CSV format file and then were visualized using Python scripts using the following Python visualization libraries:
  - **Matplotlib [36]**
  - **Seaborn[37]**
  - **Pandas [38]**

6.2 Dask Setup

- **Python:** We installed Dask using Python's pip package manager with the command `pip install "dask[complete]"`, following the guide [39]. Dask integrates seamlessly with Python for parallel computing [40], offering scalability similar to Spark [41].
- **Terminal Interaction:** Dask scripts were executed through Python scripts in the terminal, allowing for easy integration with other Python libraries and tools.
- **Matplotlib [36]:** We used `Matplotlib`, a versatile plotting library in Python, to generate visual representations of our data. It allowed us to create static, interactive, and animated visualizations, enabling us to effectively communicate insights from the data.
- **Seaborn[37]:** A library for making statistical graphics in Python. Seaborn [37] is built on top of Matplotlib, it provides a higher-level API, which makes it easier to create attractive and informative statistical graphics. On the other hand, Matplotlib [36] offers more flexibility for customizing plots.
- **Pandas [38]:** For data manipulation and preprocessing, we utilized Pandas . This powerful library provided us with easy-to-use data structures like DataFrames, enabling efficient data wrangling. Pandas' functionality, such as grouping, filtering, and reshaping data, was instrumental in preparing our datasets for analysis and visualization.

7 Cloud Setup

This subsection details setting up Apache Spark [35] and Dask [39]on GCP [32] for the data analysis. We first started by creating three Virtual Machines (VMs)[42], with one serving as the master node, and the other two as worker nodes.

Instances Specifications:

- Operating System: Debian GNU/Linux 12 (bookworm), version 12
- Operating System type: 64-bit
- Instance Type: e2-medium
  - 2 Virtual Central Processing Units (vCPU)
  - 2 GB of memory

We created a single Google Cloud Storage (GCS) bucket [15] for both the input dataset and output files in CSV format for the visualization.

7.1 Apache Spark Setup

After creating the instances and setting them up by configuring the slaves file on the master node, which contained the IP addresses of all worker nodes. This setup enabled the master node to coordinate tasks and communicate with each worker in the cluster. We started by downloading and installing the following:

- **Java:** OpenJDK version 8 (Java 1.8) [33] on all nodes, as a prerequisite for running Apache Spark.
- **Scala:** Apache Spark [35] version 3.5.3 with Hadoop 3 compatibility, and Scala 2.12.18 on all nodes.
- **Spark Shell:** Interaction with Spark through Spark Shell and directly from the command line inside of the master node was done by the following command:

```
$SPARK_HOME/bin/spark-shell \
--master spark://10.128.0.2:7077 \
--jars /home/nouarif4/Downloads/gcs-connector-hadoop2-latest.jar \
```

which specified the path for Spark in the master node, followed by specifying which internal IP address is the master node's, and a GCS connector for accessing the dataset bucket.

- **Visualization of Analysis**

for visualization of Spark's analysis, we wrote Python scripts using the libraries Matplotlib[36], Seaborn [37], and Pandas [38].

7.2 Dask Setup

For our Dask cluster, we only had to use **Python** and its package manager **pip** for the setup. Starting by installing Dask in each node using

```
pip3 install dask[distributed] dask[dataframe]
```

then we Installed Visualization Libraries

```
 pip3 install matplotlib seaborn plotly
```

lastly a library that allows Dask to interact with Google Cloud Storage

```
pip install gcsfs
```

Then, we initiated the Dask scheduler on the master node using the command:

```
dask scheduler --port 8786 --dashboard-address :8787
```

On each worker node, we connected to the master scheduler using:

```
dask worker tcp://<master-ip>:8786
```

This established the Dask cluster, linking worker nodes to the master node for coordinated processing.

For visualization of Dask's analysis, we used the same Python libraries we used to visualize Spark, Matplotlib[36], Seaborn [37], and Pandas [38].

8 Initial Data Analysis on Local Machine

In this section, we will address our business questions outlined in Section 3 by using both Apache Spark and Dask. By applying these two frameworks, we aim not only to derive insights from the data but also to evaluate these frameworks.

8.1 Data Preprocessing

We prepared and cleaned the data for Spark and Dask by loading the eCommerce dataset, stored in Parquet format. Data cleaning involves (Drop rows with null values, remove unnecessary columns, Remove duplicate rows) The key columns used for analysis include:

- `event_type`: Defines the type of event. We only have cart (addToCart, we will be using both names interchangeably), view, and purchase
- `category_code`: Product category
- `price`: Product price
- `brand`: Brand of the product

Both systems successfully loaded and read the dataset for the subsequent analyses.

8.2 Answering Our Business Questions

In this section, we analyze the dataset to answer our business questions 3. The following questions guide our analysis:

Question 1: Which product categories have the highest conversion rates?

To determine the conversion rates, we will analyze the data by counting the number of purchases and views for each product category and calculating the conversion rate using :

$$\mathbf{ConversionRate = (TotalAddToCartEvents\ /\ TotalViews)}$$

Spark

Below is the result of the product conversion rates analysis in Scala using Spark. Figure **??** shows all product categories, while Figure 5 shows only the top 20 results.

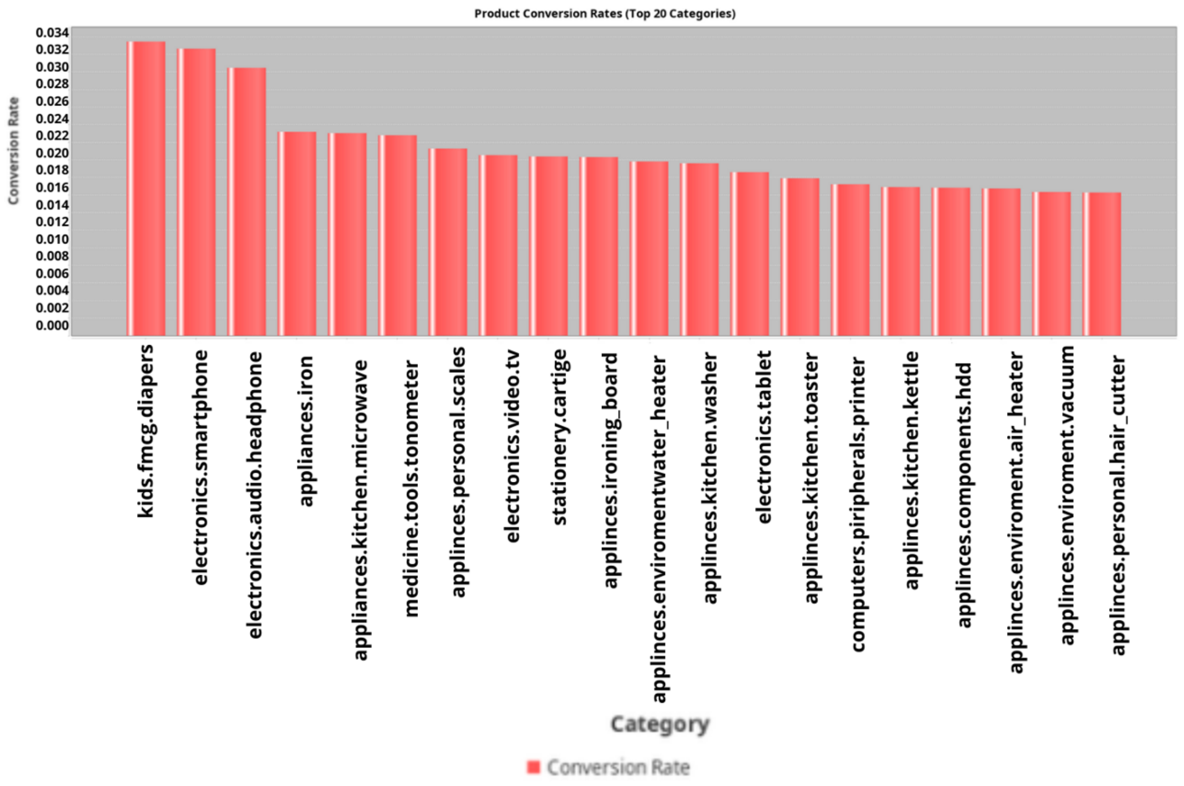


Figure 5: Bar chart showing the highest 20 product conversion rates and the corresponding categories. We can see that kids' diapers have the highest conversion rate(viewing and adding it to the cart).

Dask

Below are the results of the conversion rate analysis in Dask. Figure 6 shows the top 20 results.

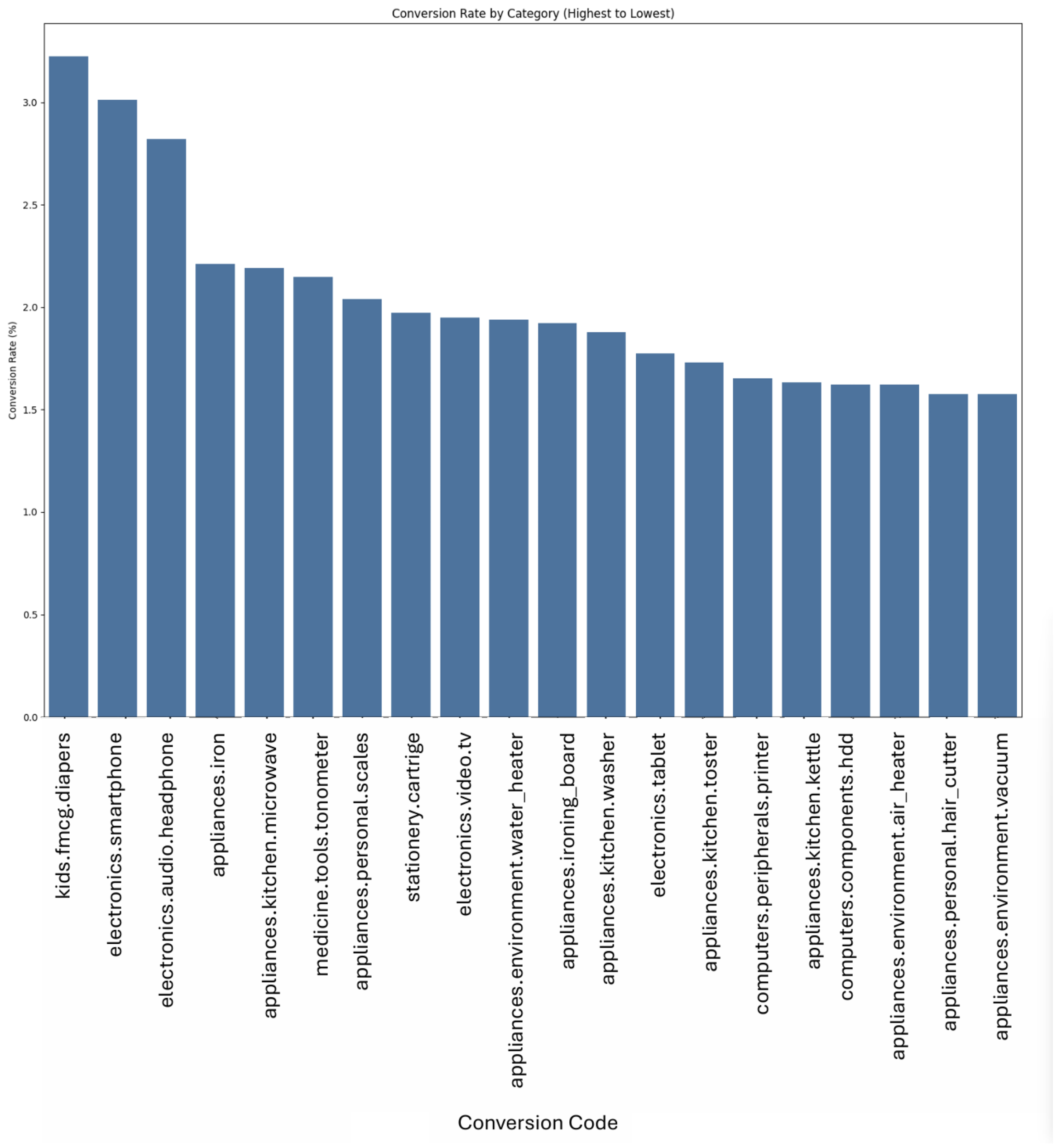


Figure 6: A bar chart showing the highest 20 product conversion rates and the corresponding categories. Kids' diapers have the highest conversion rate.

From the above figures, we can see that the two systems have the same results.

Question 2: How do product price ranges within each category affect customer engagement (product viewing or adding items to the cart) and conversion rates?

first, we needed to group the similar categories (more specifically, the categories that have the same first part before the dot, as per our dataset's category code column). Then, we categorized products into specific price ranges. After that, the dataset was filtered by event types (`view` and `cart`), and aggregation operations were performed to calculate:

- **Total Views**: The total number of views for each combination of price range and category groups.
- **Total Cart Events**: The total number of add-to-cart events for the same combinations.

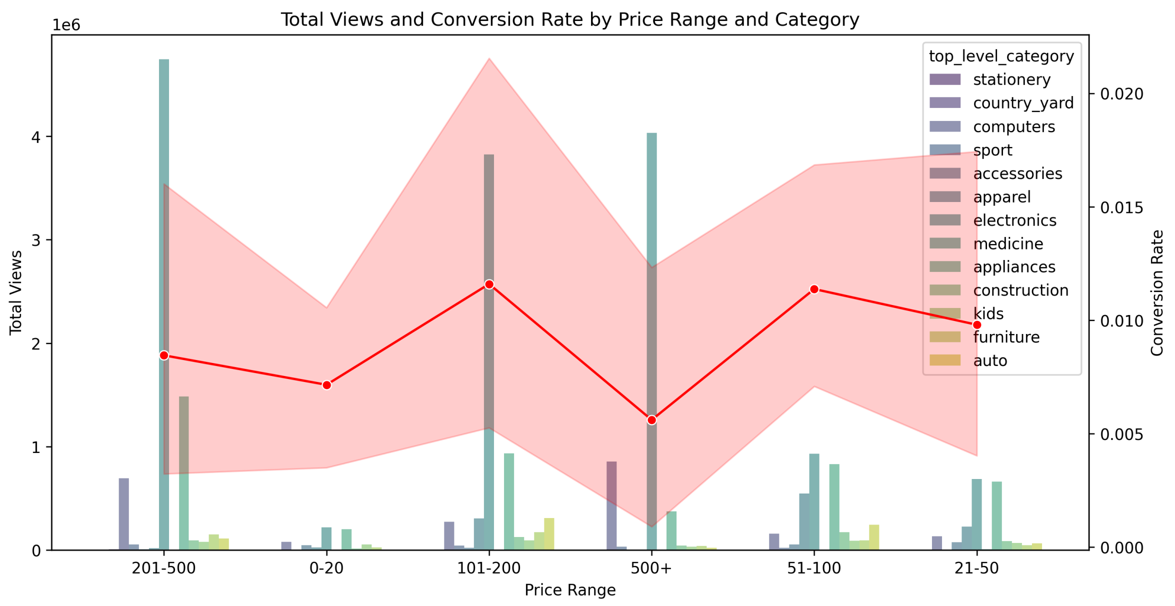


Figure 7: The chart shows Spark's result of how much the price ranges of the categories affect customer engagements.

- **Conversion Rate:** This is computed by dividing the total add-to-cart events by the total views for each price range and category combination.

A bar plot is created to visually represent total views per price range, with bars color-coded by top-level category, and a line plot overlays the bar plot to depict conversion rates for each price range.

Approach:

1. **Segment Products by Price Range**: Categorize products into price ranges.
2. **Aggregate Engagement Metrics**: Calculate total views and add-to-cart actions for each price range within each category.
3. **Compute Conversion Rates**: Determine conversion rates for each price range.
4. **Visualize the Data**: A dual-axis plot shows total engagements and conversion rates across price ranges.

Spark's result in Figure **??** shows that the middle part has the highest conversion rate and lower conversion rate when the prices are lower, but it has different results regarding the high prices.

Dask's result

The next Figure shows that the middle part of the chart has the highest conversion rate. And at the edges, it gets lower. Even though the price ranges are lower at the beginning, the conversion rate is low. We assume that it is because people make assumptions about the quality of a product or item based on its price. The lower the price, the poorer the perceived quality. But on the other hand, when the price gets higher, people tend to avoid buying the product because they have cheaper options. Which means the middle part has the highest conversion rate.

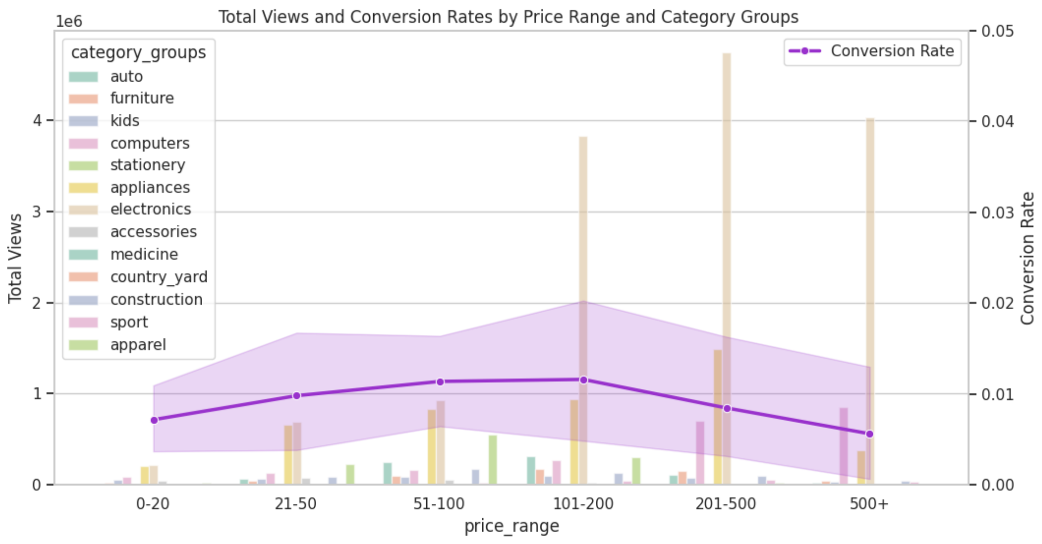


Figure 8: The chart shows Dask's result of how much the price ranges of the categories affect customer engagement.

The highlighted margin around the conversion rate line is known as a confidence interval or error band. It visually represents the uncertainty or variability in the conversion rate estimates at each price range. It provides a range of values that is likely to contain the true conversion rate for a given price range. It helps to show how reliable your conversion rate estimate is. A narrow band indicates high certainty, while a wider band suggests greater variability in the estimates [43].

9 Data Analysis on Cloud

Our previous analysis was done locally, in this section we will show our analysis on-cloud, specifically in GCP [42].

Question 1: Which product categories have the highest conversion rates?
Spark's result:

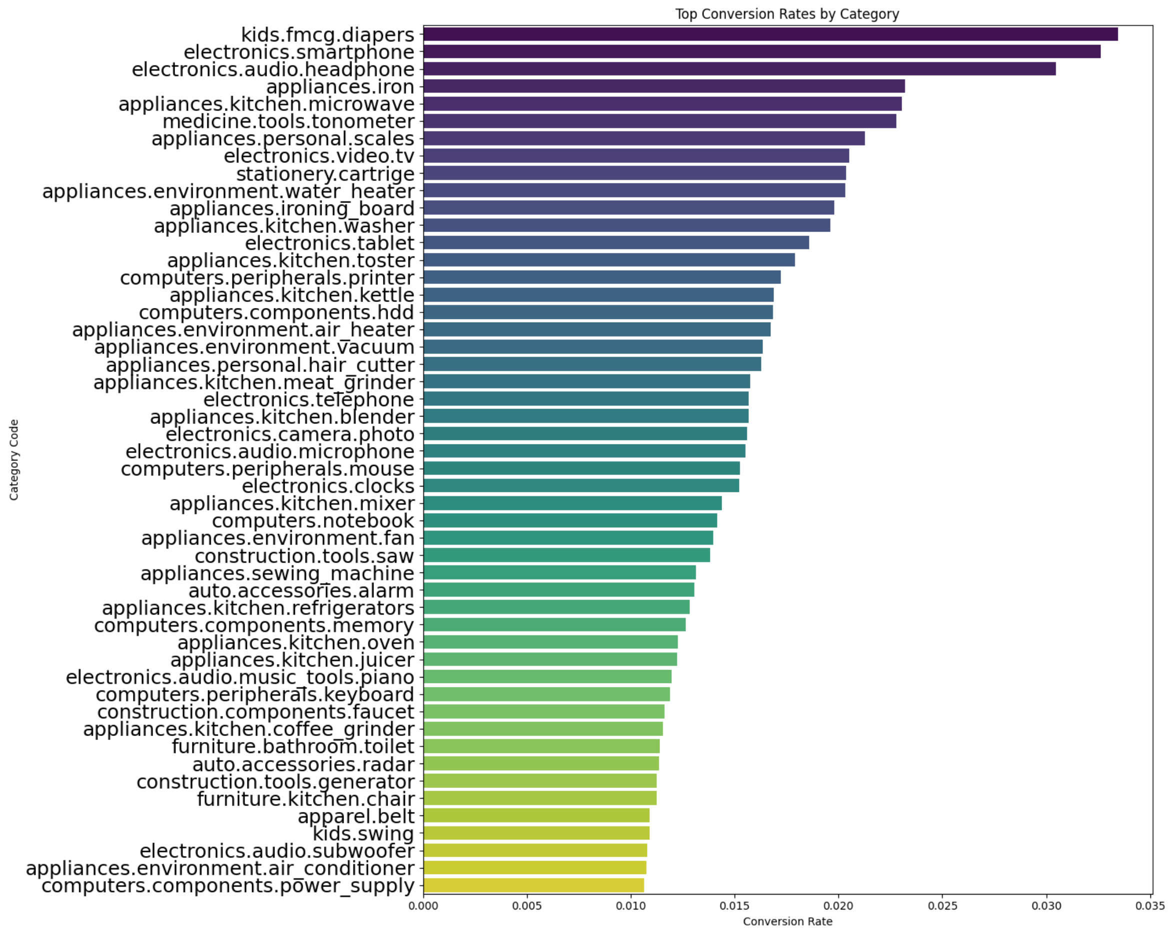


Figure 9: A bar chart showing top 20 categories in conversion rate

Dask's result:

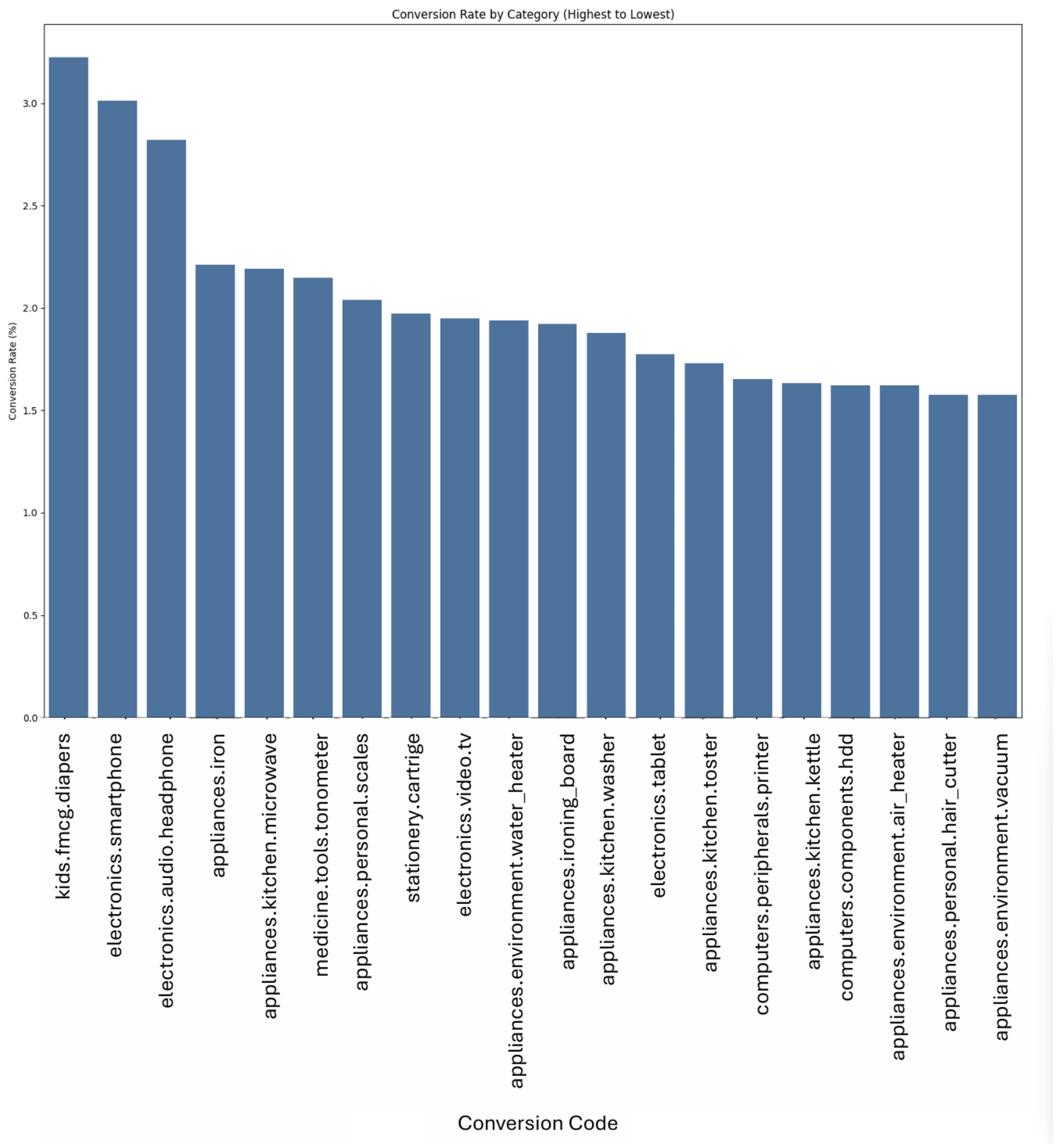


Figure 10: A bar chart showing the top 20 categories in conversion rate

Question 2: How do product price ranges within each category affect customer engagement (product viewing or adding items to the cart) and conversion rates?

Spark's result:

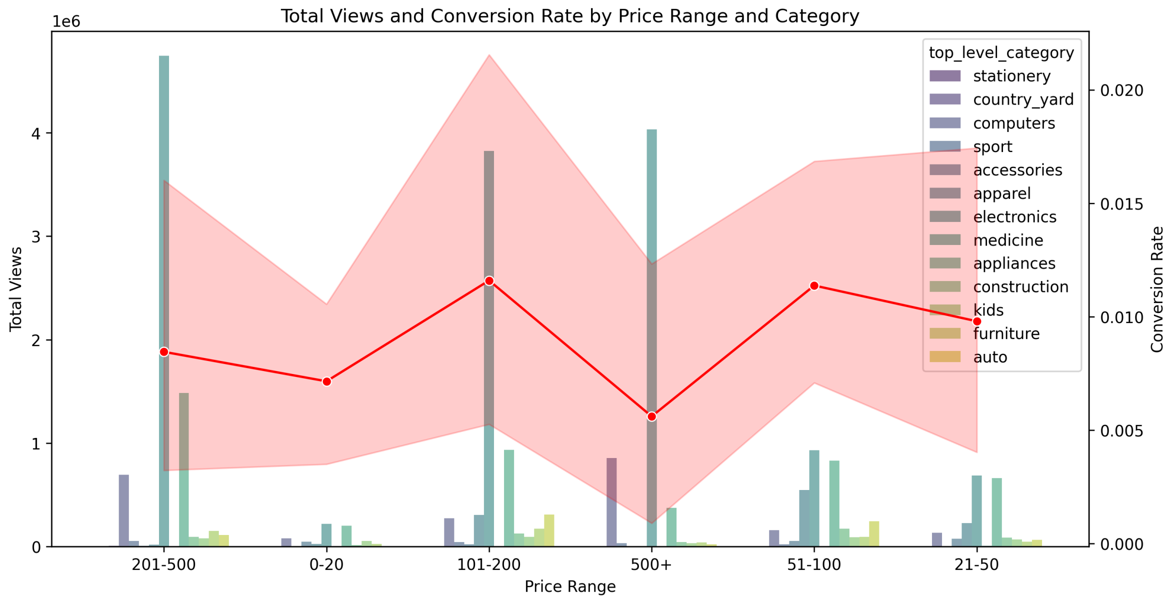


Figure 11: A chart showing the total views and conversion rates by price range and category

Dask's result:

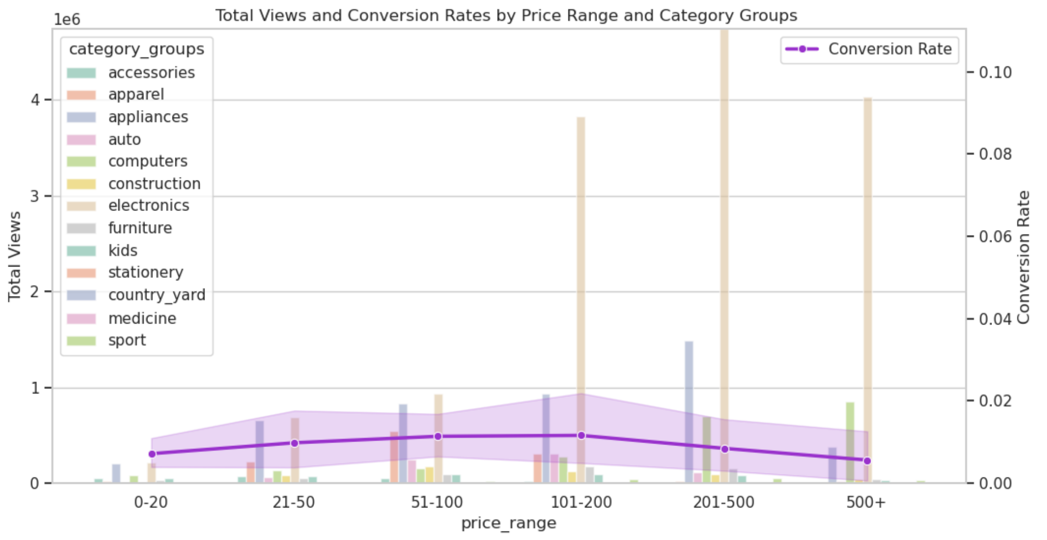


Figure 12: A chart showing the total views and conversion rates by the price ranges and categories

9.1 Performance of Both Clusters

After the analysis in Spark [17] and Dask [19] on the Google Cloud platform [32], we obtained the results needed to perform the primary goal of the evaluation paper, which is to evaluate and compare Spark and Dask systems on a cloud platform.
Firstly, we needed to gather data from GCP's monitoring tools [44], to track and evaluate using specific evaluation metrics. We focused on crucial resource utilization metrics that provide insights into how efficiently each framework manages system resources.

9.1.1 CPU utilization [45]:

CPU utilization is essential to determine the cloud infrastructure's efficiency and performance. It provides insight into how effectively Spark and Dask use available processing resources. CPU utilization has an impact on request latency and performance. As CPU utilization increases, more tasks compete for processing resources, reducing the overall performance. This metric will help us identify which processing system utilizes resources effectively.

9.1.2 CPU utilization during Q1:

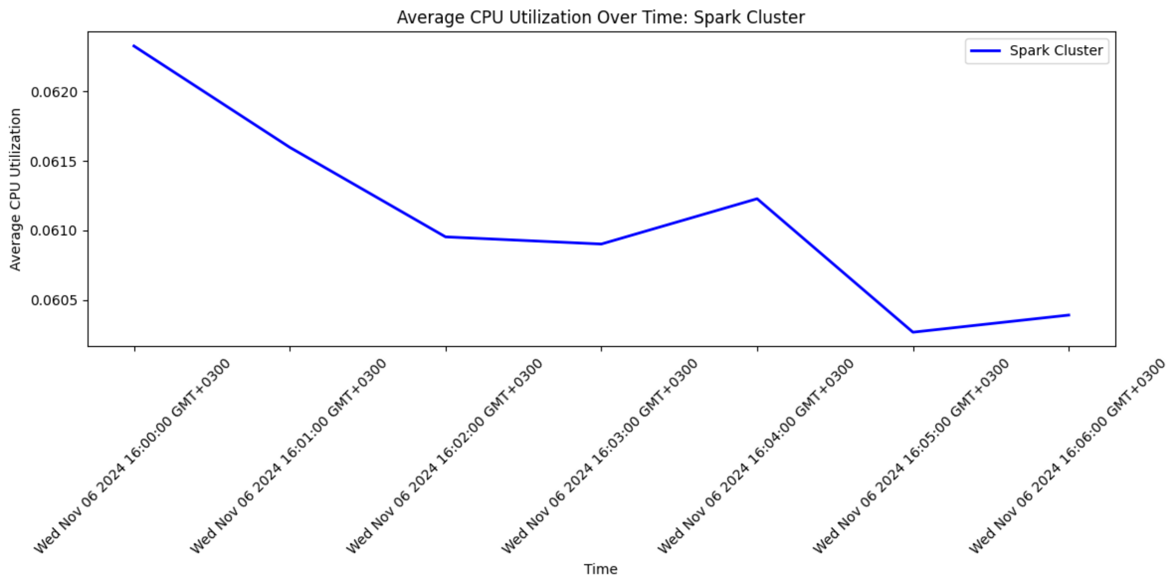


Figure 13: Average CPU Utilization of The Spark Cluster During The Analysis of Question 1

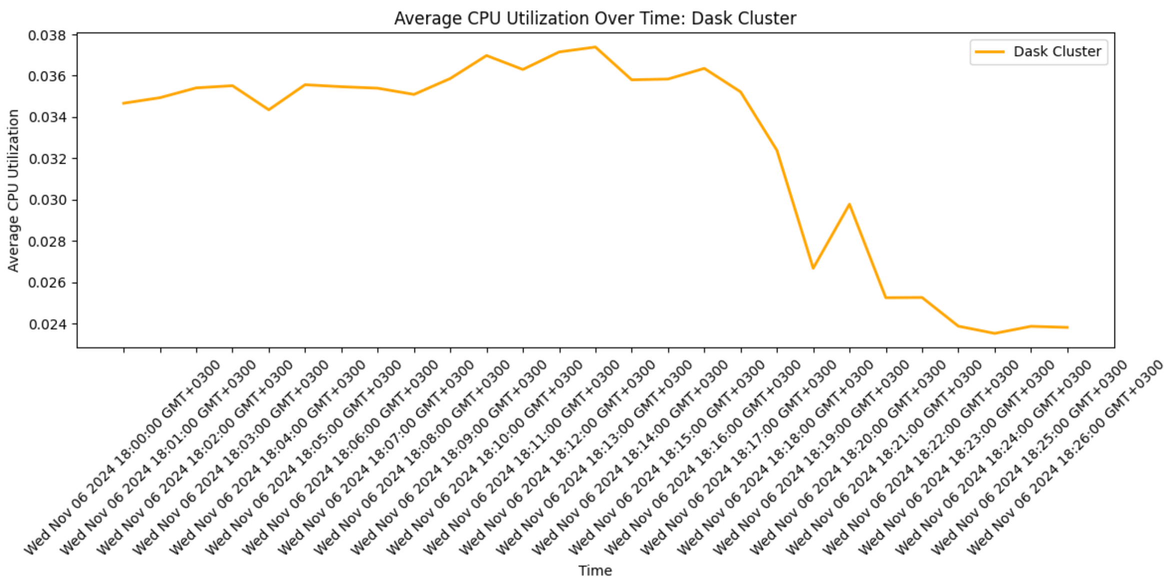


Figure 14: Average CPU Utilization of The Dask Clustering During The Analysis of Question 1

Dask total CPU utilization during Q1: 0.104000442
Spark total CPU utilization during Q1: 0.186980424

9.1.3 Conclusion:

Spark's higher CPU utilization may lead to better performance for compute-intensive tasks, but it comes at the cost of higher resource consumption. This makes Spark a good choice for large-scale, compute-heavy workloads where performance is critical, while Dask's lower CPU utilization makes it a more cost-effective option, especially for workloads that are not compute-intensive or require efficient resource usage which makes it well-suited for smaller-scale or distributed workloads where cost and efficiency are more important than raw performance.

9.1.4 CPU Utilization during Q2:

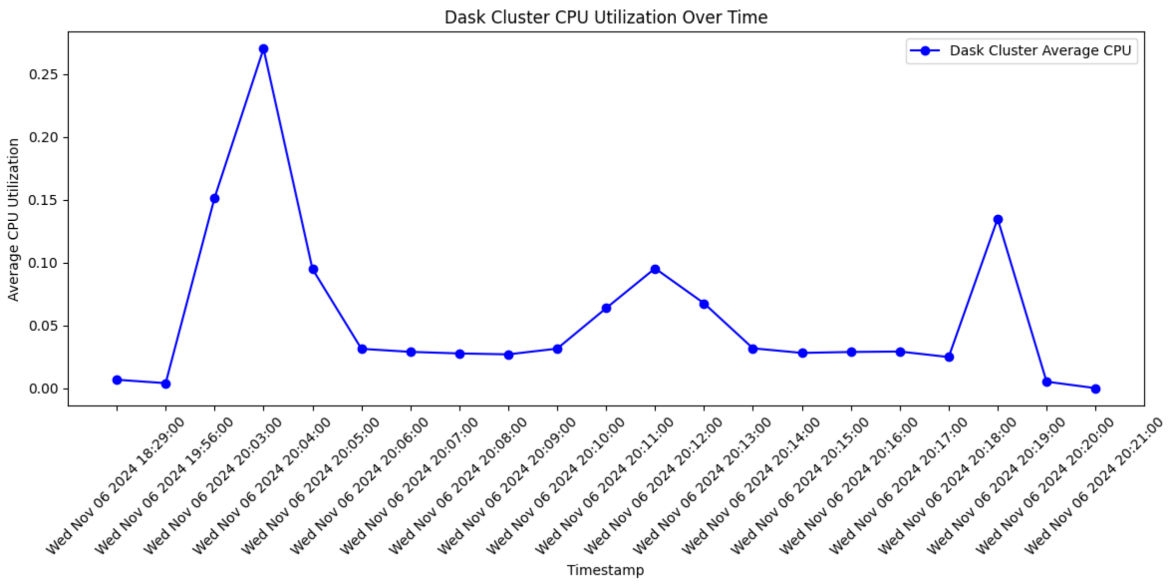


Figure 15: CPU utilization of the spark cluster in the analysis of question 2.

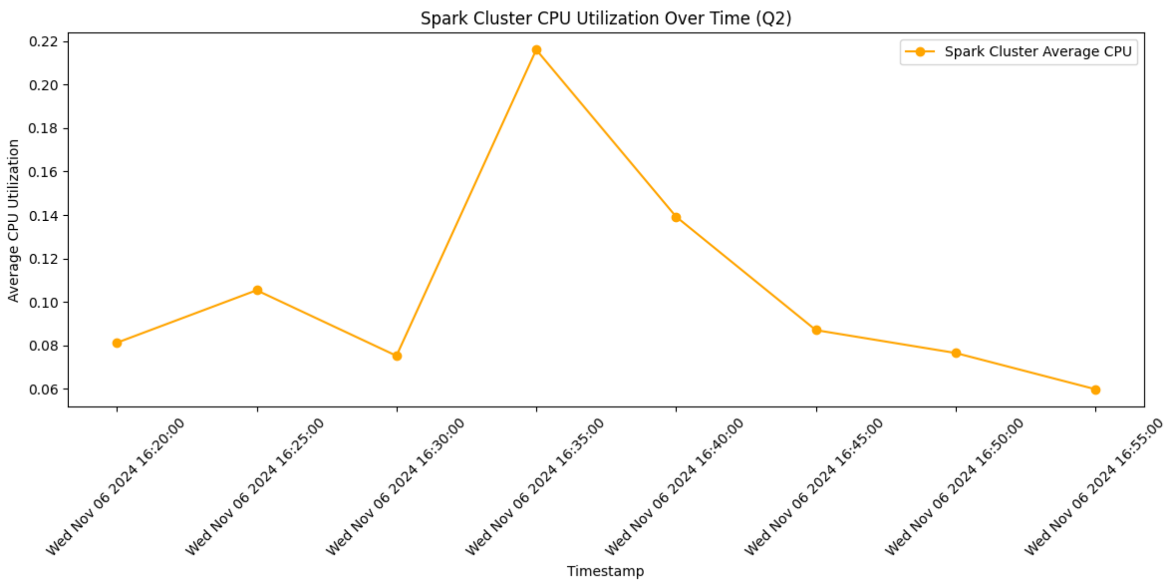


Figure 16: Enter Caption

Dask total CPU utilization during Q2: 1.1833073666716483
Spark total CPU utilization during Q2: 0.8406969770095442

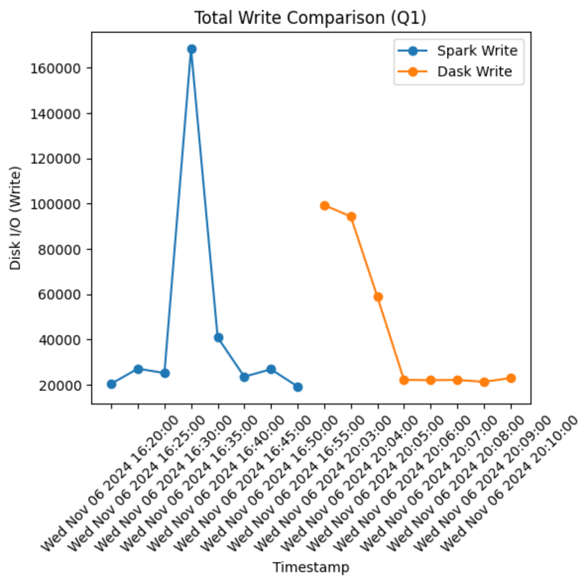
9.1.5 Conclusion:

Dask is more efficient in terms of CPU utilization, making it suitable for real-time, memory-intensive, or lightweight workloads. Its low and consistent CPU usage ensures predictable performance and minimal resource overhead.
Spark is better suited for batch processing, large-scale data transformations, or CPU-intensive workloads. Its higher CPU utilization and variability make it more appropriate for environments with ample CPU resources.

9.1.6 Disk I/O [46]:

Disk I/O operations determine the performance of the block storage volume attached to the virtual machine. The higher the disk I/O operations, the higher the cost and performance. We monitored the disk I/o operations done by both Spark and Dask to analyze which system offers high performance and lower cost.

9.1.7 Disk I/O size during Q1:



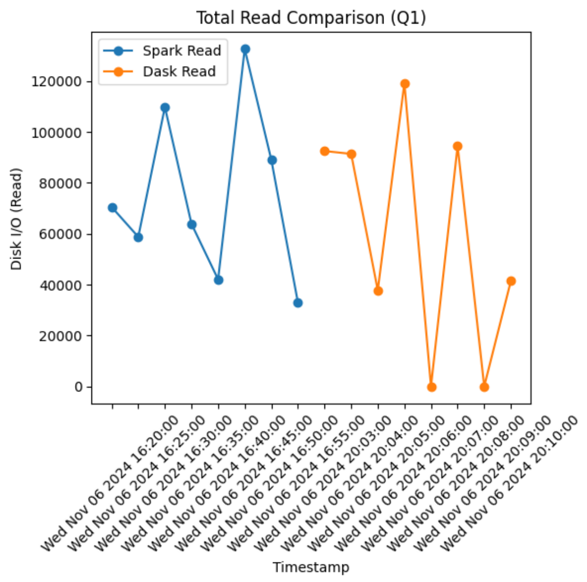


Figure 17: Total Write and Total Read of Spark and Dask Answering Question 1

Spark total write during Q1: 351681.52999999997
Spark total read during Q1: 599479.22
Dask total write during Q1: 363277.36000000004
Dask total read during Q1: 476741.8

9.1.8 Conclusion:

Dask is more efficient and consistent in terms of disk I/O, making it a better choice for disk-sensitive workloads while Spark is more disk-intensive but may be better for large-scale data processing tasks that can handle higher disk I/O.

9.1.9 Disk I/O size during Q2:

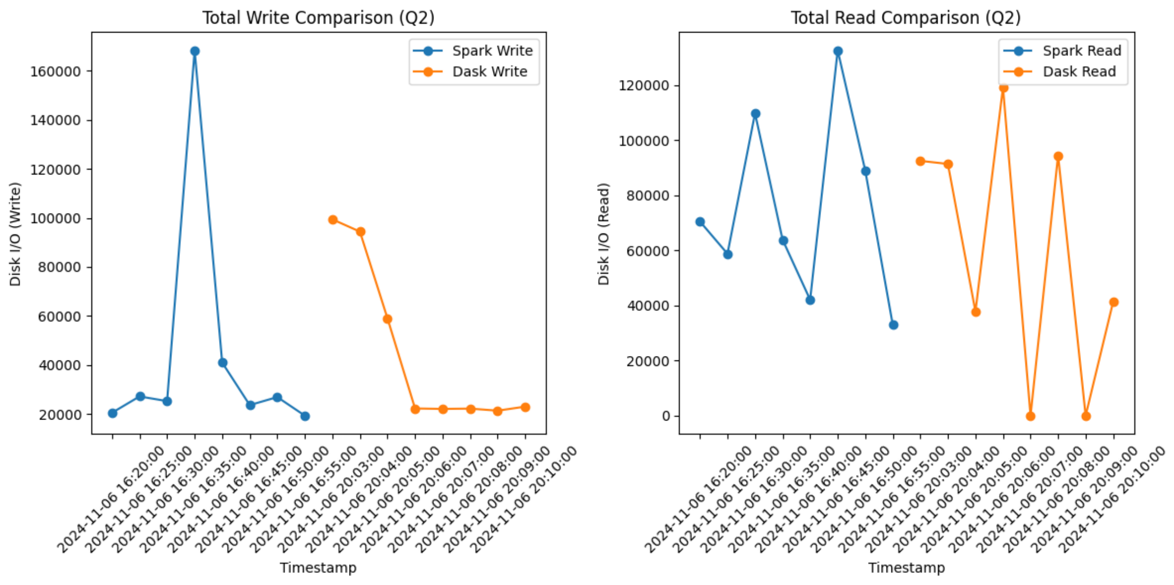


Figure 18: Total Write and Total Read of Spark and Dask Answering Question 2

9.1.10 Conclusion:

Dask appears to be more efficient in terms of disk I/O since it performs fewer read operations than Spark, which could indicate better memory utilization or caching mechanisms. Lower disk I/O generally translates to faster performance in scenarios where disk access is a bottleneck.

9.2 Conclusion of The Performance of Both Clusters:

Based on the analysis of **CPU utilization** and **disk I/O** during Q1 and Q2, the following key findings summarize the performance characteristics of Spark and Dask:

9.2.1 CPU Utilization

- **Spark**:
  - Spark exhibits **higher CPU utilization**, which makes it well-suited for compute-intensive workloads such as batch processing, large-scale data transformations, and CPU-heavy tasks [47].
  - However, this comes at the cost of higher resource consumption, making it less efficient for lightweight or real-time workloads.
  - The variability in Spark's CPU usage suggests that it is optimized for environments with ample CPU resources.
- **Dask**:
  - Dask demonstrates **lower and more consistent CPU utilization**, making it a more cost-effective and resource-efficient option.
  - It is better suited for real-time, memory-intensive, or lightweight workloads, where predictable performance and minimal resource overhead are critical [48].
  - Dask's efficient CPU usage makes it ideal for smaller-scale or distributed workloads where cost and efficiency are prioritized over raw performance.

9.2.2 Disk I/O

- **Spark**:
  - Spark is **more disk-intensive**, which can be beneficial for large-scale data processing tasks that require significant disk operations [47].
  - However, its higher disk I/O may lead to performance bottlenecks in disk-sensitive workloads, making it less efficient in scenarios where disk access is a limiting factor.
- **Dask**:
  - Dask is **more efficient and consistent** in terms of disk I/O, performing fewer read operations compared to Spark.
  - This indicates better memory utilization and caching mechanisms, which reduce reliance on disk operations [48].
  - Dask's lower disk I/O makes it a better choice for disk-sensitive workloads, ensuring faster performance in scenarios where disk access is a bottleneck.

9.2.3 Recommendations

- **When to Choose Spark**:
  - For large-scale, compute-intensive workloads that require high performance and can tolerate higher resource consumption.
  - For batch processing or data transformation tasks where higher CPU and disk I/O are acceptable.
  - In environments with ample CPU and disk resources.
- **When to Choose Dask**:
  - For real-time, memory-intensive, or lightweight workloads that require efficient resource usage and predictable performance.
  - For disk-sensitive workloads where lower disk I/O is critical for performance.
  - In cost-sensitive environments where resource efficiency is a priority.

9.2.4 Conclusion

- Spark is optimized for performance in large-scale, compute-heavy, and disk-intensive workloads, but it comes at the cost of higher resource consumption.
- Dask is more efficient in terms of both CPU and disk I/O, making it a better choice for cost-effective, real-time, and disk-sensitive workloads.

10 Challenges in Distributed Data Processing for Business Analytics

While working with both Dask and Apache Spark, we encountered several challenges. One significant issue was the performance of Dask, which we found to be slower than Spark. This led to longer processing times than expected. Dask's architecture, though flexible, can sometimes struggle with larger datasets, particularly when compared to Spark, which is designed for high-speed cluster computing.

For data visualization, we used Matplotlib [36], Seaborn [37], and Pandas [38] for Dask since the analysis was in Python too. For Spark, we initially planned to use **Vegas**[49], which is a Scala library for creating visualizations, but it did not work as expected due to compatibility issues with our dataset format. Instead, we switched to JFreeChart [50], a Java-based library. Although JFreeChart is primarily used with Java, we successfully implemented it with Scala due to its cross-language support. The results of JFreeChart charts were lacking on the UI front, which in our on-cloud analysis made us switch and use the same Python libraries we used in Dask's visualization, by saving the result of Spark's analysis in a CSV format file and then using it in a Python script with the visualization libraries, Matplotlib [36], Seaborn [37], and Pandas [38].

Unfortunately, we have been affected by the lack of the system's educational resources, and even if there are available resources, they are outdated, especially since systems are developing very quickly, but we overcame this problem by constantly referring to the system's documentation. Moreover, we found it challenging to interact with the systems through the terminal, as any command typed could not be modified later. This added to the complexity of running tasks, especially when debugging or fine-tuning the code became necessary.

References

[1] Amazon Web Services. Cloud computing services - amazon web services (aws). https://aws.amazon.com/, n.d. Accessed: 11 October 2024.

[2] Common-canvas. Commoncanvas-s-c. https://huggingface.co/common-canvas/CommonCanvas-S-C, n.d. Accessed: 11 September 2024.

[3] Open Data Platfom. Open data platfom. https://od.data.gov.sa/en, n.d. Accessed: 11 September 2024.

[4] F. Seddik. Arabic viral tweets (). Kaggle, 2022. https://www.kaggle.com/datasets/fahdseddik/arabic-viral-tweets Accessed: 11 September 2024.

[5] Your machine learning and data science community. Kaggle, n.d. https://www.kaggle.com/ Accessed: 11 September 2024.

[6] 2.5+ million rows egyptian datasets collection. Kaggle, n.d. https://www.kaggle.com/datasets/mostafanofal/two-million-rows-egyptian-datasets Accessed: 11 September 2024.

[7] H. Sayed. Egypt - arabic political 600k tweets. Kaggle, 2019. https://www.kaggle.com/datasets/hazemshokry/egypt-arabic-politics-tweets Accessed: 11 September 2024.

[8] M. Kechinov. Ecommerce behavior data from multi category store, December 9 2019. Kaggle.

[9] M. Kechinov. Ecommerce behavior data from multi category store. Kaggle, 2019. https://www.kaggle.com/datasets/mkechinov/ecommerce-behavior-data-from-multi-category-store Accessed: 11 September 2024.

[10] Rees46 open cdp - open source customer data platform. REES46, n.d. https://rees46.com/en/open-cdp Accessed: 11 September 2024.

[11] Drive more revenue with niche-specific personalization engine. REES46, n.d. https://rees46.com/ Accessed: 11 September 2024.

[12] Project jupyter. https://jupyter.org/, n.d. Accessed: 11 September 2024.

[13] Author Name. Title of the paper. *Journal Name*, 2024.

[14] Amazon s3 console. https://console.aws.amazon.com/S3/, n.d. Accessed: 11 September 2024.

[15] Google. Cloud storage. https://cloud.google.com/storage?hl=ar#object-storage-for-companies-of-all-sizes, n.d.-a. Accessed: 2024-11-06.

[16] A. Lee. How to ask the right questions as a data scientist. Medium, 2022. https://towardsdatascience.com/how-to-ask-the-right-questions-as-a-data-scientist-913621907411 Accessed: 11 September 2024.

[17] Apache Spark^{TM}. Apache Spark^{TM} - Unified Engine for Large-Scale Data Analytics. https://spark.apache.org/, n.d. Accessed: 2024-10-17.

[18] Apache hadoop. https://hadoop.apache.org/, n.d. Accessed: 11 September 2024.

[19] Powered by. Dask, n.d. Available at: https://www.dask.org/powered-by. Accessed: 11 September 2024.

[20] ClickHouse. What is clickhouse? https://clickhouse.com/docs/en/, 2024. Accessed: 12 October 2024.

[21] Yandex. Clickhouse: An open-source columnar database management system. https://clickhouse.com/docs/en/history/, 2024. Accessed: 12 October 2024.

[22] John Doe. Database selection for big data analytics: A comparative study. *Data Systems Journal*, 12(3):45–59, 2021.

[23] Evgenia Kuzmenko. Why it's time to move on from hadoop. Kitrum Blog, January 2024. https://kitrum.com/blog/why-its-time-to-move-on-from-hadoop/ Accessed: 11 September 2024.

[24] Apache spark: Pros and cons. AltexSoft Blog, July 2023. https://www.altexsoft.com/blog/apache-spark-pros-cons/ Accessed: 11 September 2024.

[25] NumPy Developers. Numpy: The fundamental package for array computing with python. https://numpy.org/about/, n.d. Accessed: 2024-10-17.

[26] Domino. Spark, dask, ray: Choosing the right framework. Domino Blog, September 2024. Available at: https://domino.ai/blog/spark-dask-ray-choosing-the-right-framework. Accessed: 11 September 2024.

[27] Apache spark vs dask vs pandas. Censius Blog, September 2024. Available at: https://censius.ai/blogs/apache-spark-vs-dask-vs-pandas. Accessed: 11 September 2024.

[28] Amazon Web Services. Amazon ec2 - scalable cloud computing services. https://aws.amazon.com/ec2/, 2023. Accessed: 11 October 2024.

[29] Amazon Web Services. Amazon redshift - fast, simple, cost-effective data warehousing. https://aws.amazon.com/redshift/, 2023. Accessed: 11 October 2024.

[30] AWS Documentation. What is iam? - aws identity and access management. https://docs.aws.amazon.com/IAM/latest/UserGuide/introduction.html, 2024. Accessed: 2024-11-06.

[31] Amazon Web Services. Aws product and service pricing — amazon web services. https://aws.amazon.com/pricing/, 2024. Accessed: 2024-11-06.

[32] What is google cloud platform (gcp)? Pluralsight, n.d. Available at: https://www.pluralsight.com/resources/blog/cloud/what-is-google-cloud-platform-gcp. Accessed: 10 September 2024.

[33] OpenJDK. Openjdk: Installation guide. https://openjdk.org/install/, 2024. Accessed: 12 October 2024.

[34] Sehun Kim. Comparison of languages supported in apache spark. https://sehun.me/comparison-of-languages-supported-in-apache-spark-120dc774230e, 2024. Accessed: 12 October 2024.

[35] Apache Software Foundation. Download apache spark. https://spark.apache.org/downloads.html, 2024. Accessed: 12 October 2024.

[36] Matplotlib Developers. Matplotlib: Python plotting. https://matplotlib.org/, 2024. Accessed: 2024-10-16.

[37] M. Waskom and the Seaborn Developers. Seaborn: Statistical data visualization. https://seaborn.pydata.org/, 2024. Accessed: 2024-10-16.

[38] Pandas Developers. Pandas documentation. https://pandas.pydata.org/, 2024. Accessed: 2024-10-16.

[39] Dask. Dask installation - dask documentation. https://docs.dask.org/en/stable/install.html. Accessed: 2024-10-17.

[40] Dask Developers. Dask: A flexible library for parallel computing in python. https://docs.dask.org/en/stable/, 2024. Accessed: 15 October 2024.

[41] Apache Software Foundation. Apache spark: A unified analytics engine for big data. https://spark.apache.org/, 2024. Accessed: 15 October 2024.

[42] Google. Virtual machine instances — compute engine documentation — google cloud. https://cloud.google.com/compute/docs/instances, 2024. Accessed: 2024-11-06.

[43] US Census Bureau. A basic explanation of confidence intervals. https://www.census.gov/programs-surveys/saipe/guidance/confidence-intervals.html, May 2017. Accessed: 2024-10-17.

[44] Google. Monitor instances with cloud monitoring — spanner — google cloud. https://cloud.google.com/spanner/docs/monitoring-cloud, n.d. Accessed: 2024-11-06.

[45] Google. Cpu utilization metrics — spanner — google cloud. https://cloud.google.com/spanner/docs/cpu-utilization, n.d. Accessed: 2024-11-06.

[46] Google. Configure disks to meet performance requirements — compute engine documentation — google cloud. https://cloud.google.com/compute/docs/disks/performance, n.d. Accessed: 2024-11-06.

[47] Apache Spark Documentation^{TM}. Cluster mode overview. https://spark.apache.org/docs/latest/cluster-overview.html, n.d. Accessed: 2024-12-4.

[48] Matthew Rocklin. Dask: Parallel computation with blocked algorithms and task scheduling, 2015.

[49] G. Jen. Data visualization with vegas viz and scala with spark ml. *Medium*, March 16 2020.

[50] JFreeChart Developers. Jfreechart: Open source java chart library. https://www.jfree.org/jfreechart/, 2024. Accessed: 2024-10-16.