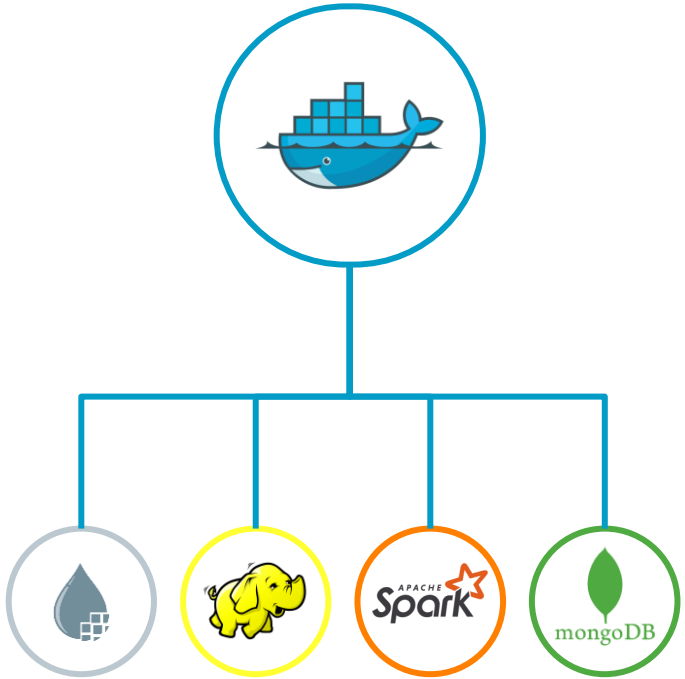
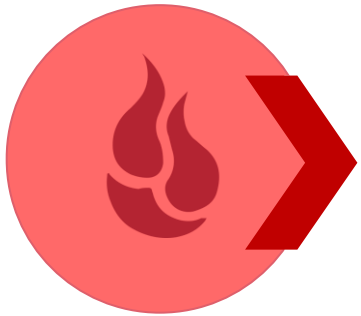
SABD – Project 1

Pieraldo Santurro - 0353517



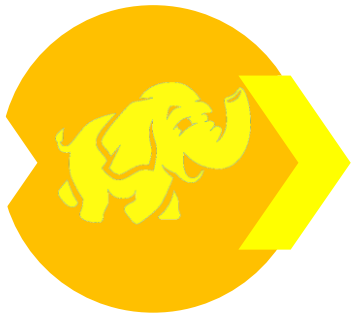
Introduzione

Obiettivi



Analisi di dati forniti da Blackblaze relativi a fallimenti di Hard Disk

Upload



Conversione in formato Parquet e caricamento su HDFS

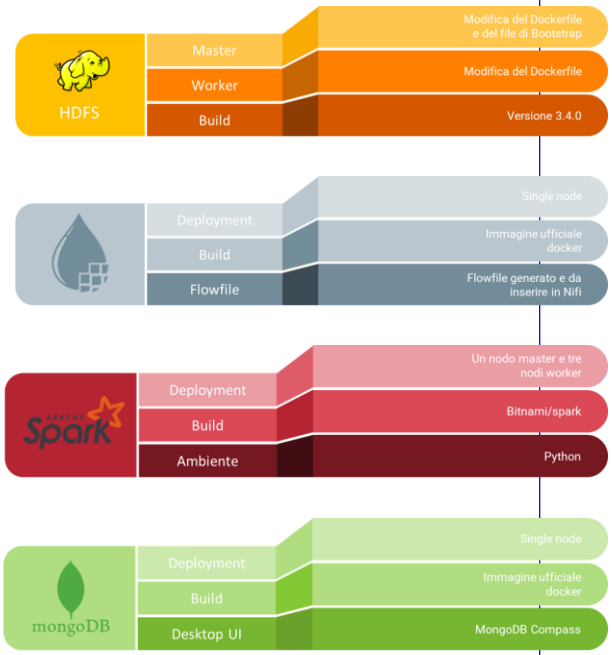
Esecuzione

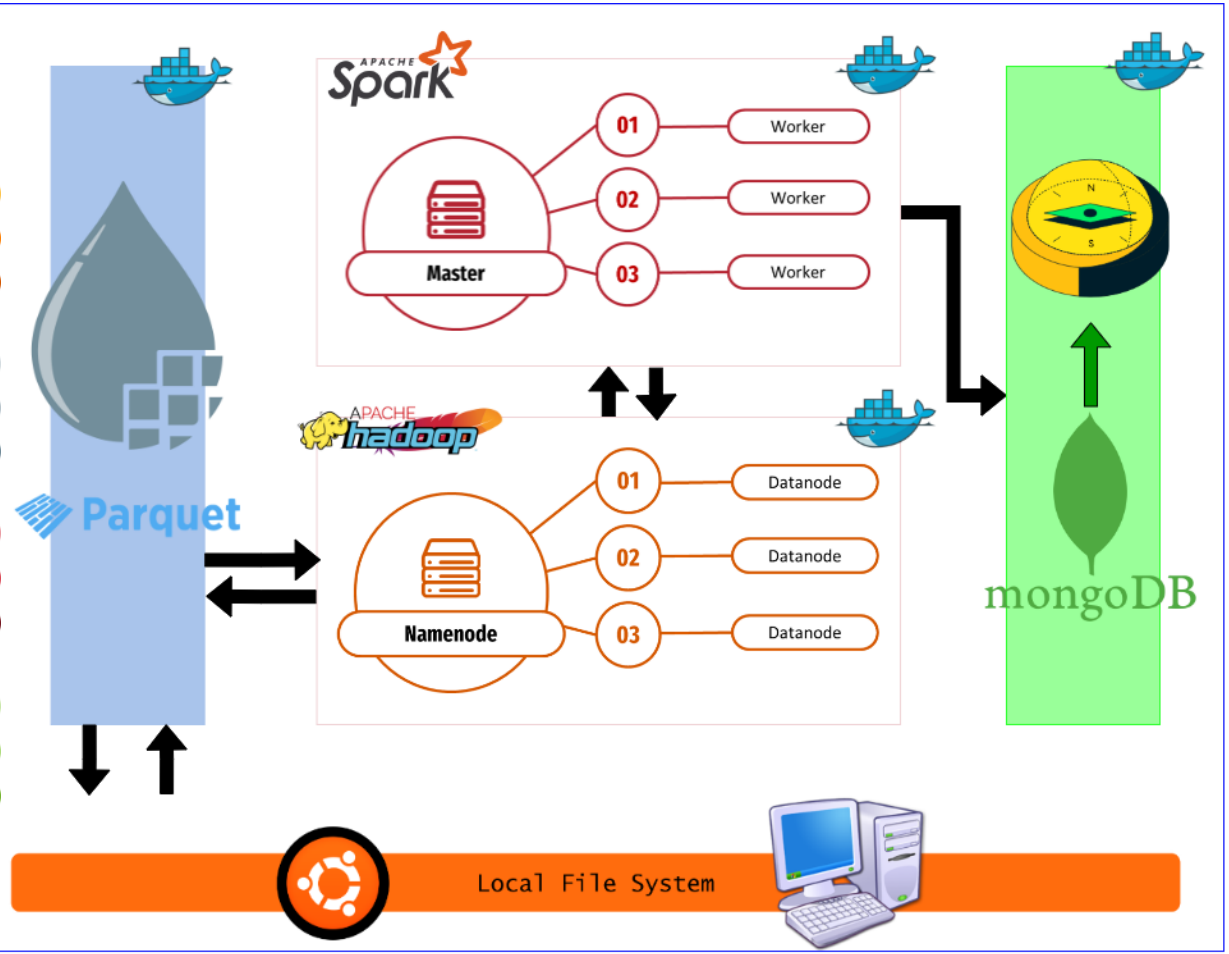
Esecuzione delle Query su Apache Spark con Dataframes API e Spark SQL Api

Download

Download dei risultati su File System locale e Upload su MongoDB

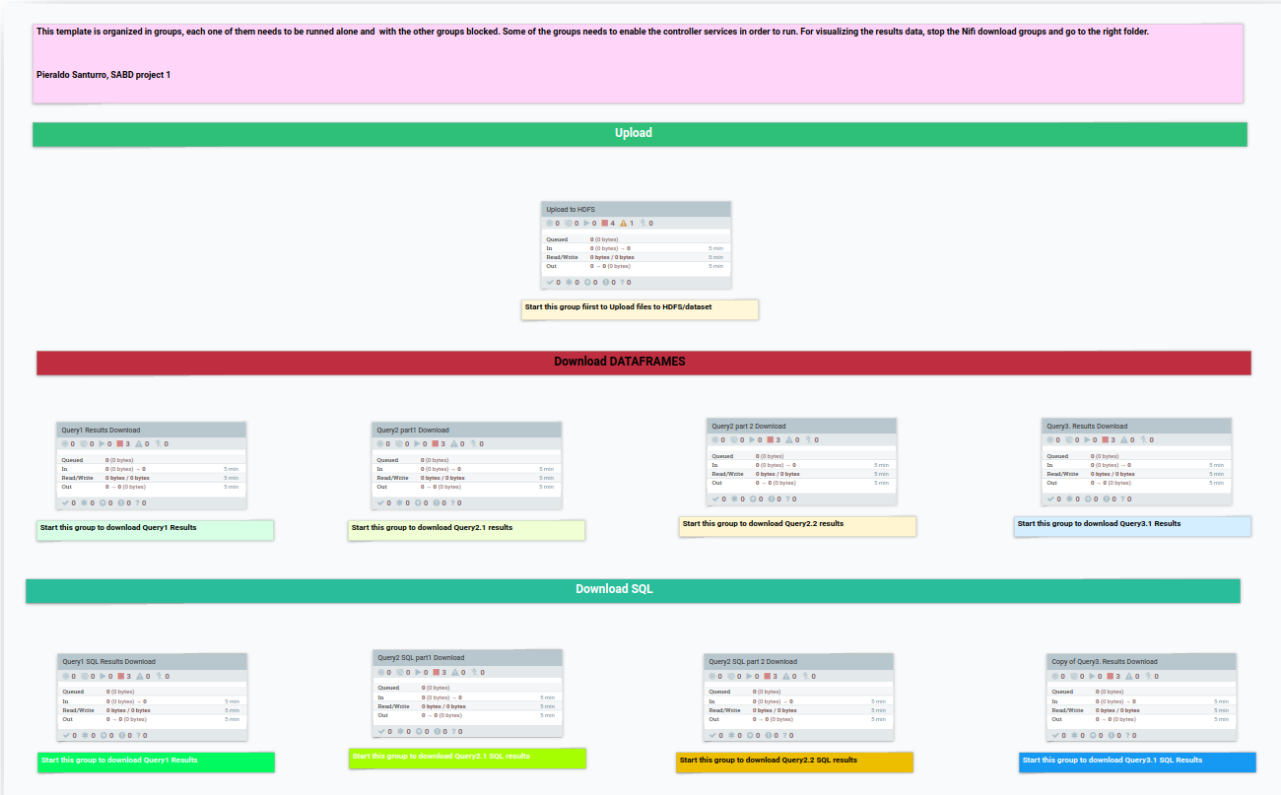
Architettura





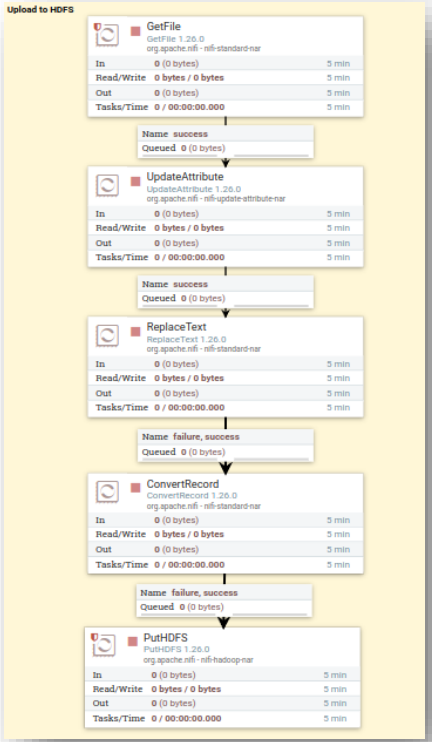
Architettura

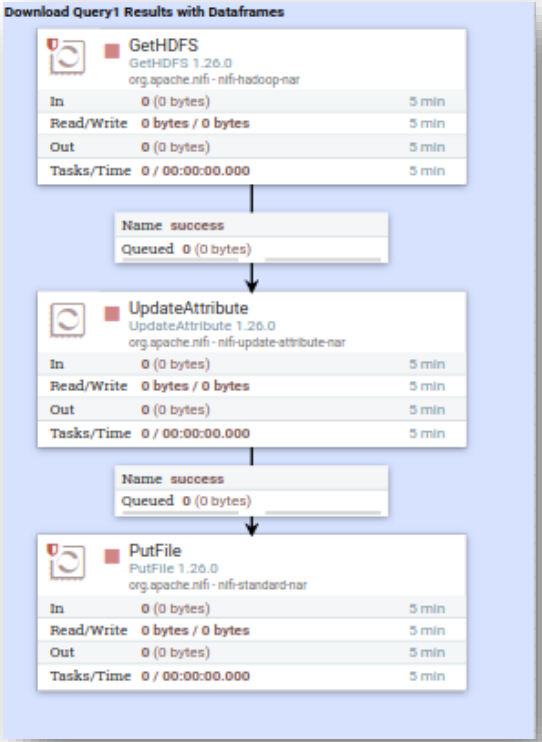
Apache Nifi



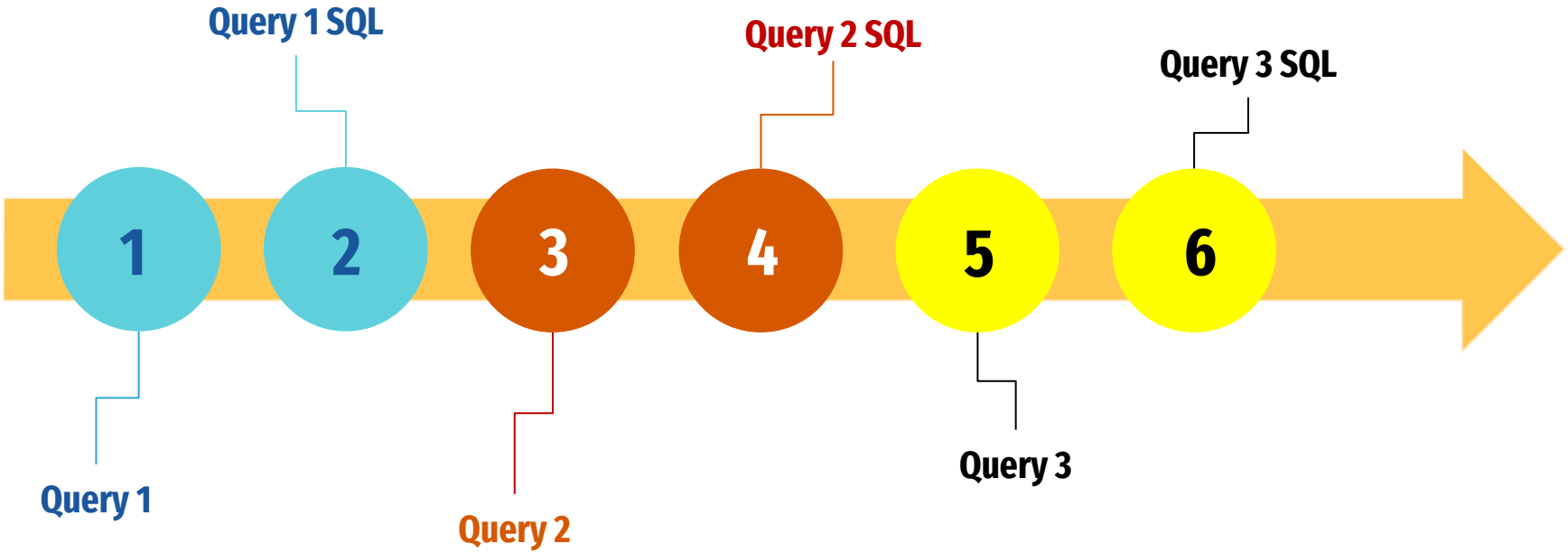
Architettura

Apache Nifi





Query


Query 1 SQL
Query 2 SQL
Query 3 SQL
1
2
3
4
5
6
Query 1
Query 2
Query 3

Query 1

Per ogni giorno, per ogni vault (si faccia riferimento al campo vault id), calcolare il numero totale di fallimenti. Determinare la lista di vault che hanno subito esattamente 4, 3 e 2 fallimenti.

Query 1

1.1 Implementazione Dataframes

format_date
Funzione per aggiungere una colonna con data formattata

filter
Filtraggio dei vault

```python
# Filtra solo i failure
failures_df = df.filter(col("failure") == 1)

# Aggiunge una colonna con la data in formato "dd-MM-yyyy"
@udf(StringType())
def format_date(date_str):
    return datetime.strptime(date_str, '%Y-%m-%dT%H:%M:%S.%f').strftime('%d-%m-%Y')

failures_df = failures_df.withColumn("date", format_date(col("date").cast("string")))

# Conta i failure per ogni giorno e per ogni vault
failures_count_df = failures_df.groupBy("date", "vault_id").count()
failures_count_df = failures_count_df.withColumnRenamed("count", "failure_count")

if print_intermediate:
    print("Conto dei failure per ogni giorno e per ogni vault:")
    failures_count_df.show()

# Filtra i vault con esattamente 2, 3 o 4 failure
filtered_df = failures_count_df.filter(col("failure_count").isin(2, 3, 4))

if print_intermediate:
    print("Vault filtrati con 2, 3 o 4 failure:")
    filtered_df.show()

# Ordina i risultati
sorted_df = filtered_df.orderBy("date", "vault_id")
```

Query 1

1.2 Implementazione SQL

format_date

Funzione alternativa a quella in dataframes per formattare la data

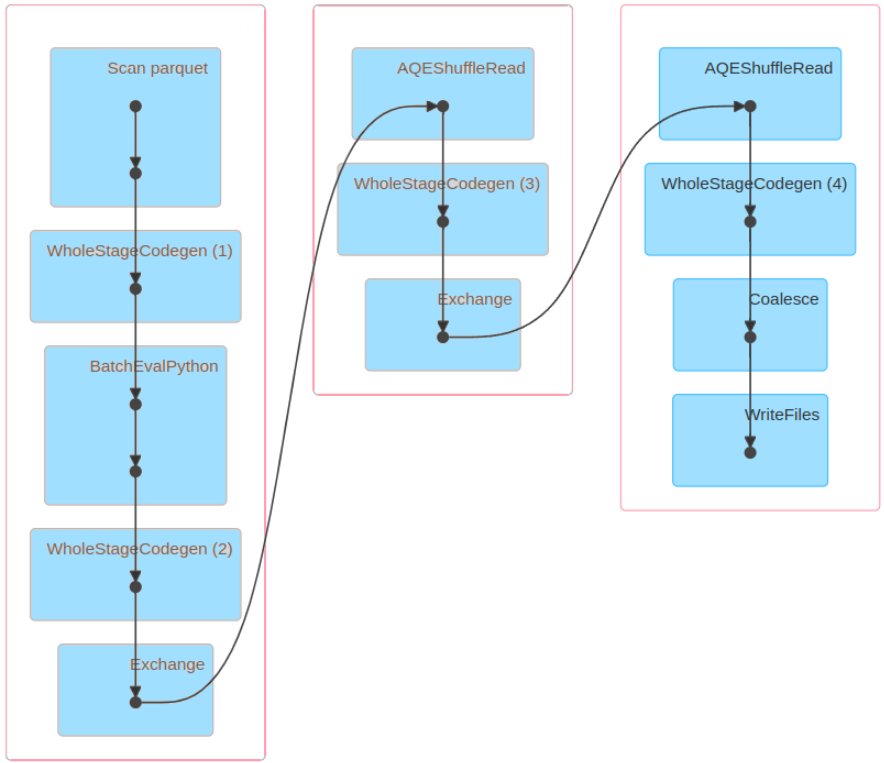
Conteggio Fallimenti

Sequenza di operazioni per conteggio dei fallimenti

```
# UDF to format date
spark.udf.register("format_date", lambda date_str: datetime.strptime(date_str, '%Y-%m-%dT%H:%M:%S.%f').strftime('%d-%m-%Y'), StringType())

# SQL query to add formatted date column
query = """
SELECT
    failure,
    vault_id,
    format_date(CAST(date AS STRING)) AS date
FROM failures
"""
formatted_failures_df = spark.sql(query)
formatted_failures_df.createOrReplaceTempView("formatted_failures")

# SQL query to count failures for each day and each vault
query = """
SELECT
    date,
    vault_id,
    COUNT(*) AS failure_count
FROM formatted_failures
GROUP BY date, vault_id
HAVING failure_count IN (2, 3, 4)
"""
filtered_df = spark.sql(query)
filtered_df.createOrReplaceTempView("filtered_failures")
```

Query 1

1.3 DAG



Query 1

1.4 Risultati

	_id ObjectId	date String	vault_id Int32	failure_count Int32
1	ObjectId('666568238049a8…	"04-04-2023"	1066	2
2	ObjectId('666568238049a8…	"04-04-2023"	1113	3
3	ObjectId('666568238049a8…	"05-04-2023"	1010	2
4	ObjectId('666568238049a8…	"05-04-2023"	1014	2
5	ObjectId('666568238049a8…	"05-04-2023"	1032	3
6	ObjectId('666568238049a8…	"05-04-2023"	1041	2
7	ObjectId('666568238049a8…	"05-04-2023"	1044	2
8	ObjectId('666568238049a8…	"05-04-2023"	1093	3
9	ObjectId('666568238049a8…	"07-04-2023"	1033	2
10	ObjectId('666568238049a8…	"07-04-2023"	1133	2
11	ObjectId('666568238049a8…	"08-04-2023"	1124	2
12	ObjectId('666568238049a8…	"08-04-2023"	1400	2
13	ObjectId('666568238049a8…	"11-04-2023"	1036	2
14	ObjectId('666568238049a8…	"11-04-2023"	1040	3
15	ObjectId('666568238049a8…	"11-04-2023"	1090	3
16	ObjectId('666568238049a8…	"11-04-2023"	1092	2
17	ObjectId('666568238049a8…	"11-04-2023"	1093	2
18	ObjectId('666568238049a8…	"12-04-2023"	1096	2
19	ObjectId('666568238049a8…	"12-04-2023"	1113	3
20	ObjectId('666568238049a8…	"14-04-2023"	1120	4

	_id ObjectId	date String	vault_id Int32	failure_count Int32
21	ObjectId('666568238049a8…	"14-04-2023"	1124	2
22	ObjectId('666568238049a8…	"14-04-2023"	1128	2
23	ObjectId('666568238049a8…	"14-04-2023"	1161	2
24	ObjectId('666568238049a8…	"15-04-2023"	1113	2
25	ObjectId('666568238049a8…	"15-04-2023"	1118	2
26	ObjectId('666568238049a8…	"17-04-2023"	1053	2
27	ObjectId('666568238049a8…	"17-04-2023"	1066	2
28	ObjectId('666568238049a8…	"17-04-2023"	1113	2
29	ObjectId('666568238049a8…	"18-04-2023"	1033	2
30	ObjectId('666568238049a8…	"18-04-2023"	1113	2
31	ObjectId('666568238049a8…	"18-04-2023"	1118	2
32	ObjectId('666568238049a8…	"19-04-2023"	1091	2
33	ObjectId('666568238049a8…	"20-04-2023"	1113	2
34	ObjectId('666568238049a8…	"20-04-2023"	1132	2
35	ObjectId('666568238049a8…	"21-04-2023"	1127	2

Query 2

Calcolare la classifica dei 10 modelli di hard disk che hanno subito il maggior numero di fallimenti. La classifica deve riportare il modello di hard disk e il numero totale di fallimenti subiti dagli hard disk di quello specifico modello. In seguito, calcolare una seconda classifica dei 10 vault che hanno registrato il maggior numero di fallimenti. Per ogni vault, riportare il numero di fallimenti e la lista (senza ripetizioni) di modelli di hark disk soggetti ad almeno un fallimento.

Query 2

2.1.1 Implementazione Dataframes

Calcolo dei fallimenti

Calcolo del numero totale dei fallimenti per ogni modello di Hard Disk

Ordinamento dei Risultati

Vengono presi solamente i primi 10 valori utili

```
# Calcola il numero totale di fallimenti per ogni modello di hard disk
failures_count = df.groupBy("model").agg(spark_sum(col("failure")).alias("failures_count"))

# Ordina i risultati in ordine decrescente e prendi solo i primi 10
sorted_failures = failures_count.orderBy(col("failures_count").desc()).limit(10)

if print_intermediate:
    print("Classifica dei modelli di hard disk con il maggior numero di fallimenti:")
    sorted_failures.show()

# Salva i risultati in HDFS in formato CSV nella directory /results2.1
sorted_failures.write.csv("hdfs://master:54310/results2.1/", header=True, mode="overwrite")
```

Scrittura dei risultati

Risultato scritto in una specifica cartella dell'HDFS, riservata al primo risultato della Query 2

Query 2

2.1.2 Implementazione Dataframes

Raggruppamento per vault

Calcolo del numero totale dei fallimenti per ogni modello di Hard Disk

Ordinamento dei Risultati

Vengono presi solamente i primi 10 valori utili

Scrittura dei risultati

Risultato scritto in una specifica cartella dell'HDFS, riservata al secondo risultato della Query 2

```python
# Raggruppa per vault_id e aggrega i risultati
vault_failures = failures_df.groupBy("vault_id") \
    .agg(
        count("failure").alias("total_failures"),
        collect_set("model").alias("unique_models")
    )

# Converte l'array di modelli unici in una stringa separata da virgole
vault_failures = vault_failures.withColumn("unique_models", concat_ws(",", col("unique_models")))

if print_intermediate:
    print("Classifica dei vault con il maggior numero di fallimenti e modelli unici:")
    vault_failures.show()

# Ordina i risultati in base al numero totale di fallimenti in ordine decrescente e prendi i primi 10
sorted_vaults = vault_failures.orderBy(col("total_failures").desc()).limit(10)

if print_intermediate:
    print("Vault ordinati per numero totale di fallimenti:")
    sorted_vaults.show()

# Salva i risultati in HDFS in formato CSV nella directory /results2.2
sorted_vaults.write.csv("hdfs://master:54310/results2.2/", header=True, mode="overwrite")
```

Query 2

2.2.1 Implementazione SQL

Classifica dei fallimenti

All'interno della stessa SELECT vengono calcolati il numero totale di fallimenti e presi i primi 10 risultati utili

Scrittura dei risultati

Risultato scritto nell'HDFS

```
# Query 1: Classifica dei modelli di hard disk con il maggior numero di fallimenti
start_query1_time = datetime.now()

query = """
SELECT
    model,
    SUM(failure) AS failures_count
FROM formatted_dataset
GROUP BY model
ORDER BY failures_count DESC
LIMIT 10
"""
sorted_failures = spark.sql(query)

if print_intermediate:
    print("Query2SQL Query1:")
    sorted_failures.show()



# Save the results to HDFS in CSV format in the directory /results2.1 (Azione 2)
sorted_failures.write.csv("hdfs://master:54310/results2.1_SQL/", header=True, mode="overwrite")
```

Query 2

2.2.2 Implementazione SQL

Classifica dei fallimenti

Selezione degli Hard Disk con il maggior numero di fallimenti raggruppati per vault

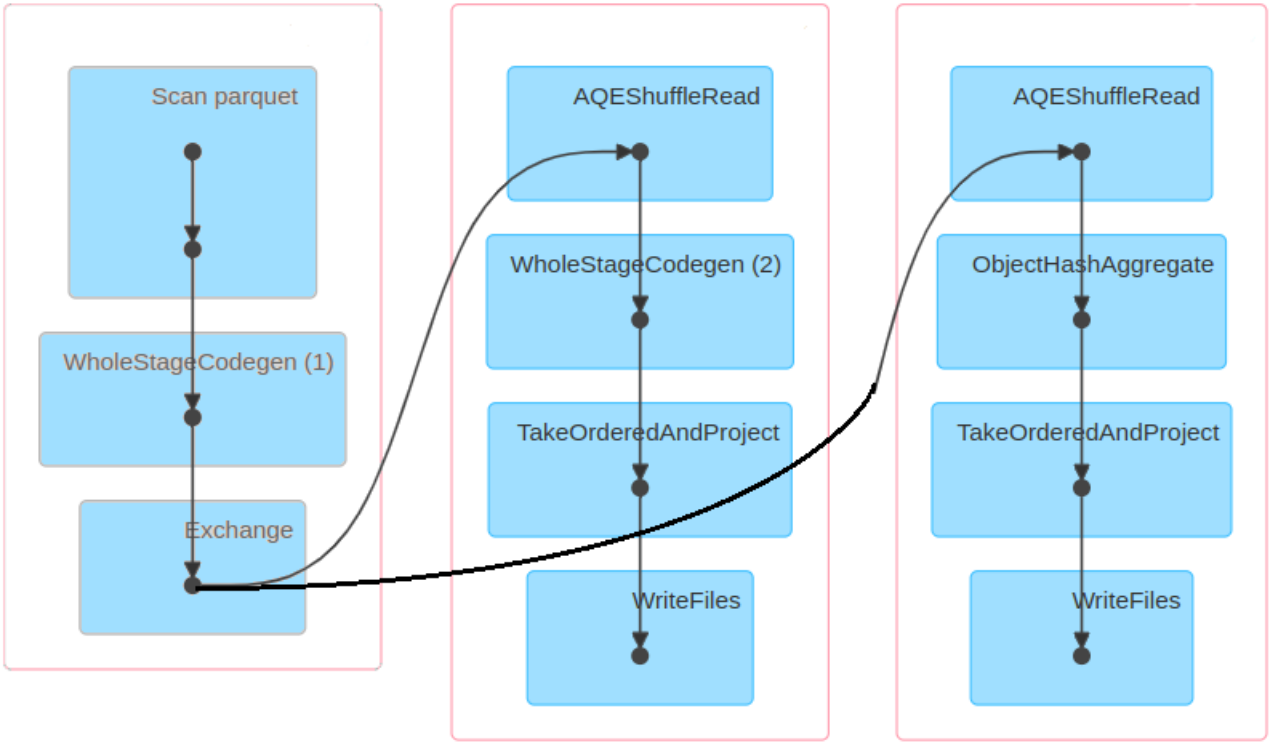
Ordinamento dei risultati

Ordinamento in ordine discendente dei primi 10 valori utili

```
query = """
SELECT
    vault_id,
    COUNT(*) AS total_failures,
    COLLECT_SET(model) AS unique_models
FROM formatted_dataset
WHERE failure = 1
GROUP BY vault_id
"""
vault_failures = spark.sql(query)
vault_failures.createOrReplaceTempView("vault_failures")

query = """
SELECT
    vault_id,
    total_failures,
    CONCAT_WS(',', unique_models) AS unique_models
FROM vault_failures
ORDER BY total_failures DESC
LIMIT 10
"""
sorted_vaults = spark.sql(query)
```

Query 2

2.3 DAG



Query 2

2.4 Risultati

	_id ObjectId	model String	failures_count Int32
1	ObjectId('6665682aed32ce…	"ST8000NM0055"	50
2	ObjectId('6665682aed32ce…	"HGST HUH721212ALN604"	48
3	ObjectId('6665682aed32ce…	"ST4000DM000"	27
4	ObjectId('6665682aed32ce…	"ST12000NM0008"	25
5	ObjectId('6665682aed32ce…	"ST8000DM002"	22
6	ObjectId('6665682aed32ce…	"TOSHIBA MG07ACA14TA"	21
7	ObjectId('6665682aed32ce…	"ST10000NM0086"	10
8	ObjectId('6665682aed32ce…	"HGST HMS5C4040BLE640"	10
9	ObjectId('6665682aed32ce…	"HGST HUH721212ALE604"	10
10	ObjectId('6665682aed32ce…	"WDC WUH721414ALE6L4"	6

	_id ObjectId	vault_id Int32	total_failures Int32	unique_models String
1	ObjectId('66656832a68ced…	1113	15	"HGST HUH721212ALN604"
2	ObjectId('66656832a68ced…	1120	10	"HGST HUH721212ALN604"
3	ObjectId('66656832a68ced…	1093	9	"ST10000NM0086"
4	ObjectId('66656832a68ced…	1118	8	"HGST HUH721212ALN604"
5	ObjectId('66656832a68ced…	1053	8	"ST8000NM0055,TOSHIBA MQ…
6	ObjectId('66656832a68ced…	1032	7	"ST8000DM002"
7	ObjectId('66656832a68ced…	1090	7	"ST8000NM0055,WDC WD5000…
8	ObjectId('66656832a68ced…	1055	6	"ST8000NM0055"
9	ObjectId('66656832a68ced…	1124	6	"HGST HUH721212ALN604,HG…
10	ObjectId('66656832a68ced…	1066	6	"TOSHIBA MG07ACA14TA"

Query 3

Calcolare il minimo, 25-esimo, 50-esimo, 75-esimo percentile e massimo delle ore di funzionamento (campo s9 power on hours) degli hark disk che hanno subito fallimenti e degli hard disk che non hanno subito fallimenti. Si presti attenzione, il campo s9 power on hours riporta un valore cumulativo, pertanto, le statistiche richieste dalla query devono far riferimento all'ultimo giorno utile di rilevazione per ogni specifico hard disk (si consideri l'uso del campo serial number). Nell'output indicare anche il numero totale di eventi utilizzati per il calcolo delle statistiche.

Query 3

3.0 Passaggi Dettagliati

1. **Inizializzazione**
   Della sessione Spark
2. **Lettura**
   Dei dati da HDFS
3. **Definizione**
   Di una finestra di ripartizione
4. **Creazione**
   Di una colonna con il numero di riga per ogni partizione al fine di contrassegnare l'ultima osservazione verificata
5. **Filtraggio**
   Delle colonne per mantenere solo le colonne contrassegnate dal flag di ultima osservazione
6. **Filtraggio**
   Per fallimenti e non fallimenti
7. **Calcolo**
   Delle percentuali
8. **Scrittura**
   Dei risultati su HDFS

Query 3

3.1.1 Implementazione Dataframes

Finestra di ripartizione
Definisce una finestra di ripartizione per operare sui sul campo serial_number, suddividendo il dato in partizioni

Aggiunta colonna
Ci consentirà di tenere traccia dell'ultima osservazione effettuata

```
# Definisci una finestra di partizione per serial_number ordinata per data in ordine decrescente

window_spec = Window.partitionBy("serial_number").orderBy(col("date").desc())

# L'API Window viene utilizzata per creare una finestra di partizione che permette di eseguire funzioni di finestra
# (come row_number, rank, dense_rank) su partizioni dei dati.
# In questo caso, partitionBy("serial_number") suddivide i dati in partizioni per ogni "serial_number"
# e orderBy(col("date").desc()) ordina ogni partizione per "date" in ordine decrescente.
# Questo consente di aggiungere un numero di riga unico a ogni riga all'interno della partizione ordinata,
# utile per operazioni come trovare l'ultima osservazione per ogni hard disk.
# Maggiori dettagli sull'uso dell'API Window sono disponibili qui: https://spark.apache.org/docs/latest/sql-ref-syntax-qry-select-window.html

# Aggiungi una colonna con il numero di riga per ogni partizione
data_with_rank = data.withColumn("rank", row_number().over(window_spec))

if print_intermediate:
    print("Dataset con colonna 'rank' aggiunta:")
    data_with_rank.show()

# Filtra per mantenere solo le righe con rank 1 (ultima osservazione per ogni hard disk)
latest_data = data_with_rank.filter(col("rank") == 1).drop("rank")
```

Filtraggio
Per mantenere solo le righe che identificano l'ultima osservazione per ogni hard disk

Query 3

3.1.2 Implementazione Dataframes

Aggiunta colonna

La colonna ‘rank’ viene aggiunta al dataframe ‘data_with_rank’ e indica la posizione di ogni riga all’interno del gruppo definiro da ‘serial_number’, ordinate per ‘date’ in ordine decrescente. Il valore 1 nella colonna rank, viene assegnato solamente alla riga con la data più recente per ciascun serial number.

serial_number	date	failure	s9_power_on_hours.member0
SN123456	2024-06-01	0	500
SN123456	2024-05-30	0	490
SN123456	2024-05-25	1	480
SN789012	2024-06-02	1	1200
SN789012	2024-05-28	1	1190
SN789012	2024-05-20	0	1180
SN345678	2024-06-03	0	800
SN345678	2024-06-01	0	790
SN345678	2024-05-29	1	780

serial_number	date	failure	s9_power_on_hours.member0	rank
SN123456	2024-06-01	0	500	1
SN123456	2024-05-30	0	490	2
SN123456	2024-05-25	1	480	3
SN789012	2024-06-02	1	1200	1
SN789012	2024-05-28	1	1190	2
SN789012	2024-05-20	0	1180	3
SN345678	2024-06-03	0	800	1
SN345678	2024-06-01	0	790	2
SN345678	2024-05-29	1	780	3

Query 3

3.1.3 Implementazione Dataframes

FIltraggio

Il datarame viene filtrato e suddiviso in due dataframes differenti in base alla colonna failure

Calcolo delle statistiche

Creazione di una funzione che mediante l'utilizzo dell'API percentile_approx ci permette di eseguire operazioni percentuali su Query, al fine di rendere il calcolo il più agevole possibile

```
# Filtra i dati per fallimenti e non fallimenti
failure_data = latest_data.filter(col("failures") == 1)
no_failure_data = latest_data.filter(col("failures") == 0)

if print_intermediate:
    print("Dati di fallimento:")
    failure_data.show()
    print("Dati senza fallimento:")
    no_failure_data.show()

# Funzione per calcolare le statistiche per un DataFrame dato
def calculate_statistics(df, failure_flag):
    # Aggrega le statistiche per il DataFrame
    stats = df.agg(
        min("s9_power_on_hours.member0").alias("min"),
        expr("percentile_approx(s9_power_on_hours.member0, 0.25)").alias("25th_percentile"),
        expr("percentile_approx(s9_power_on_hours.member0, 0.5)").alias("50th_percentile"),
        expr("percentile_approx(s9_power_on_hours.member0, 0.75)").alias("75th_percentile"),
        max("s9_power_on_hours.member0").alias("max"),
        count("s9_power_on_hours.member0").alias("count")
    ).first()
```

Query 3

3.1.4 Implementazione Dataframes

Calcolo delle statistiche

- Utilizzo la funzione ‘agg’ per aggregare le statistiche
- ‘min’ calcola il valore minimo della colonna ‘s9_power_on_hours’
- ‘percentile_approx’ calcola i percentili approssimati delle colonne di ‘s9_power_on_hours’
- ‘max’ calcola il valore massimo della colonna ‘s9_power_on_hours’
- ‘first’ viene usato per ottenere il primo ed unico risultato dell’aggregazione

```
def calculate_statistics(df, failure_flag):
    # Aggrega le statistiche per il DataFrame
    stats = df.agg(
        min("s9_power_on_hours.member0").alias("min"),
        expr("percentile_approx(s9_power_on_hours.member0, 0.25)").alias("25th_percentile"),
        expr("percentile_approx(s9_power_on_hours.member0, 0.5)").alias("50th_percentile"),
        expr("percentile_approx(s9_power_on_hours.member0, 0.75)").alias("75th_percentile"),
        max("s9_power_on_hours.member0").alias("max"),
        count("s9_power_on_hours.member0").alias("count")
    ).first()
```

Query 3

3.1.5 Implementazione Dataframes

Creazione di un Dataframe con i risultati

Viene creato un Dataframe con il risultato delle operazioni effettuate

Scrittura dei Risultati

Scrittura dei risultati sull'HDFS

```
# Calcola le statistiche per i dati di fallimento e non fallimento
failure_stats = calculate_statistics(failure_data, 1)
no_failure_stats = calculate_statistics(no_failure_data, 0)

# Crea un DataFrame con i risultati
columns = ["failure", "min", "25th_percentile", "50th_percentile", "75th_percentile", "max", "count"]
results = spark.createDataFrame([failure_stats, no_failure_stats], columns)

if print_intermediate:
    print("Statistiche calcolate:")
    results.show()

# Avvia il conteggio del tempo di salvataggio
start_save_time = time.time()

# Salva i risultati in formato CSV
results.coalesce(1).write.csv("hdfs://master:54310/results3/", header=True, mode="overwrite")
```

Query 3

3.2.1 Implementazione SQL

Partition Window e row number column

Come in Dataframes, viene definita una finestra di ripartizione e viene aggiunta una colonna che tiene traccia dell'ultima osservazione

Filtraggio

Filtraggio in base all'ultima osservazione effettuata

```
# Define a partition window by serial_number ordered by date descending and add a row number column
query = """
SELECT
    *,
    ROW_NUMBER() OVER (PARTITION BY serial_number ORDER BY date DESC) AS rank
FROM formatted_dataset
"""
data_with_rank = spark.sql(query)
data_with_rank.createOrReplaceTempView("data_with_rank")

if print_intermediate:
    print("Query3SQL colonna row number aggiunta:")
    sorted_failures.show()


# Filter to keep only rows with rank 1 (latest observation for each hard disk)
query = """
SELECT
    *
FROM data_with_rank
WHERE rank = 1
"""
latest_data = spark.sql(query)
latest_data.createOrReplaceTempView("latest_data")
```

Query 3

3.2.2 Implementazione SQL

Filtraggio

Filtraggio per failure=0 e failure1

```
# Filter data for failure and non-failure
query = """
SELECT *
FROM latest_data
WHERE failures = 1
"""
failure_data = spark.sql(query)
failure_data.createOrReplaceTempView("failure_data")

query = """
SELECT *
FROM latest_data
WHERE failures = 0
"""
no_failure_data = spark.sql(query)
no_failure_data.createOrReplaceTempView("no_failure_data")
```

Query 3

3.2.3 Implementazione SQL

Funzione per il calcolo delle statistiche

Come in Dataframes, viene definita una funzione per il calcolo delle statistiche

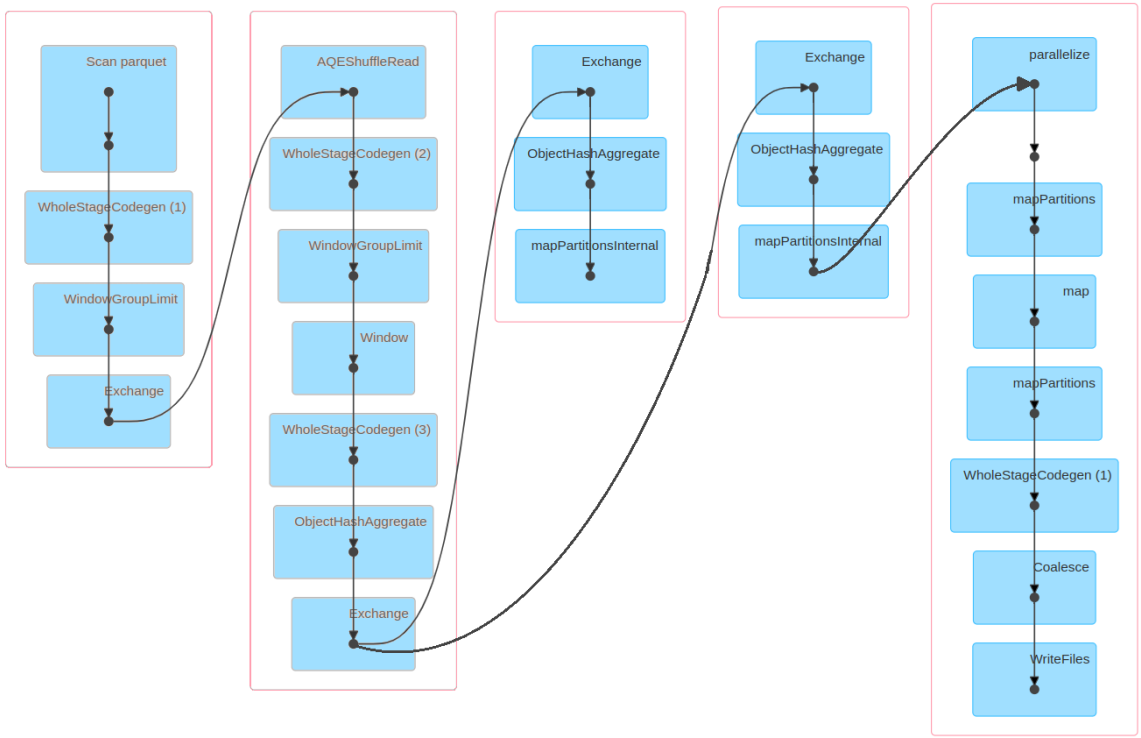
Creazione Dataframes

Dopo aver calcolato le statistiche, vengono inserite nel Dataframe risultante che verrà salvato su HDFS

```python
# Function to calculate statistics for a given DataFrame
def calculate_statistics(table_name, failure_flag):
    query = f"""
    SELECT
        {failure_flag} AS failure,
        MIN(s9_power_on_hours.member0) AS min,
        PERCENTILE_APPROX(s9_power_on_hours.member0, 0.25) AS 25th_percentile,
        PERCENTILE_APPROX(s9_power_on_hours.member0, 0.5) AS 50th_percentile,
        PERCENTILE_APPROX(s9_power_on_hours.member0, 0.75) AS 75th_percentile,
        MAX(s9_power_on_hours.member0) AS max,
        COUNT(s9_power_on_hours.member0) AS count
    FROM {table_name}
    """
    stats = spark.sql(query).first()

    statistics_list = [
        failure_flag,
        stats["min"],
        stats["25th_percentile"],
        stats["50th_percentile"],
        stats["75th_percentile"],
        stats["max"],
        stats["count"]
    ]

    return statistics_list

# Calculate statistics for failure and non-failure data
failure_stats = calculate_statistics("failure_data", 1)
no_failure_stats = calculate_statistics("no_failure_data", 0)

# Create DataFrame with the calculated statistics
columns = ["failure", "min", "25th_percentile", "50th_percentile", "75th_percentile", "max", "count"]
results = spark.createDataFrame([failure_stats, no_failure_stats], columns)
```

Query 3

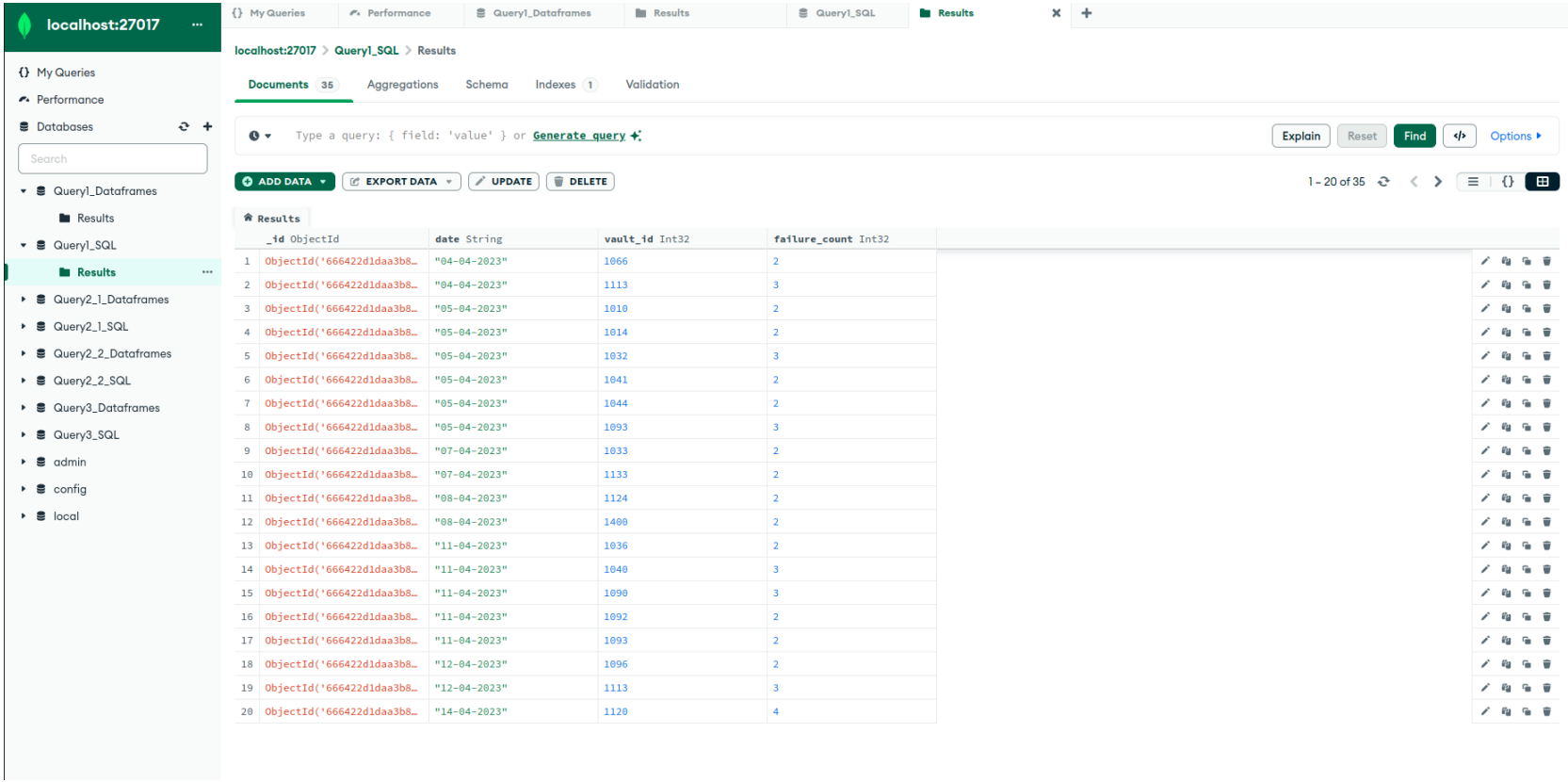
3.3 DAG



Query 3

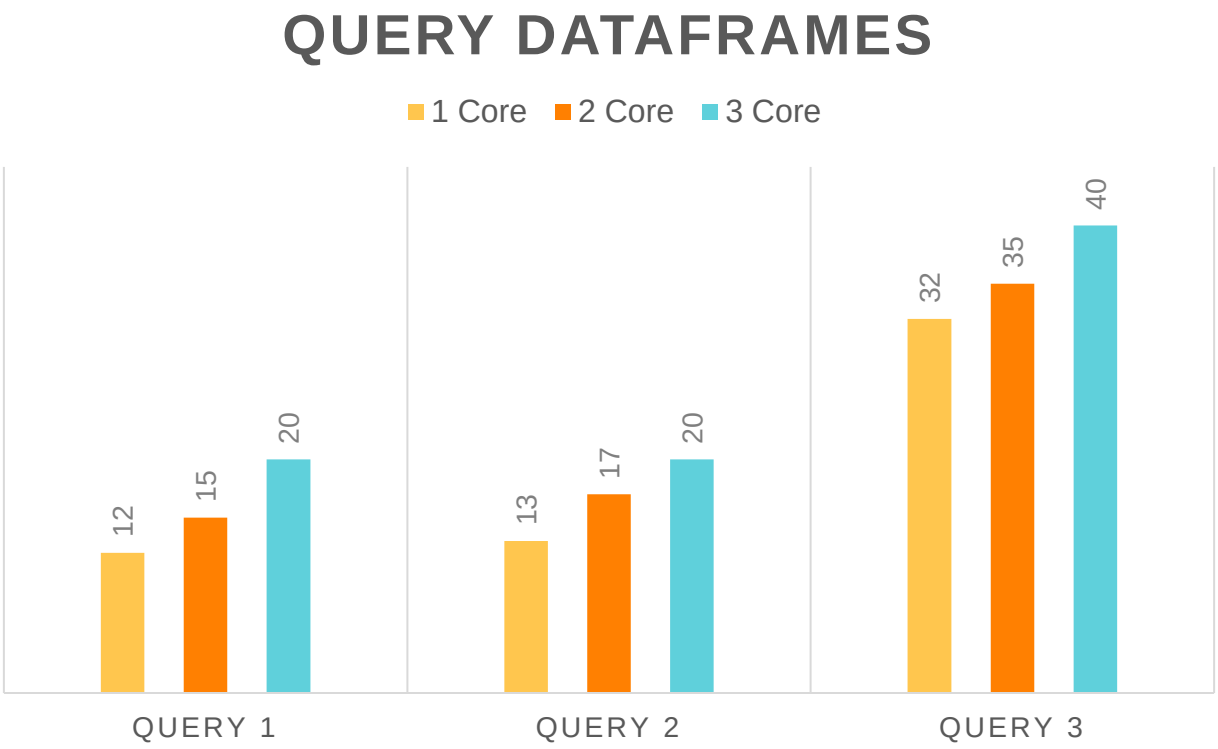
3.4 Risultati

MongoDB

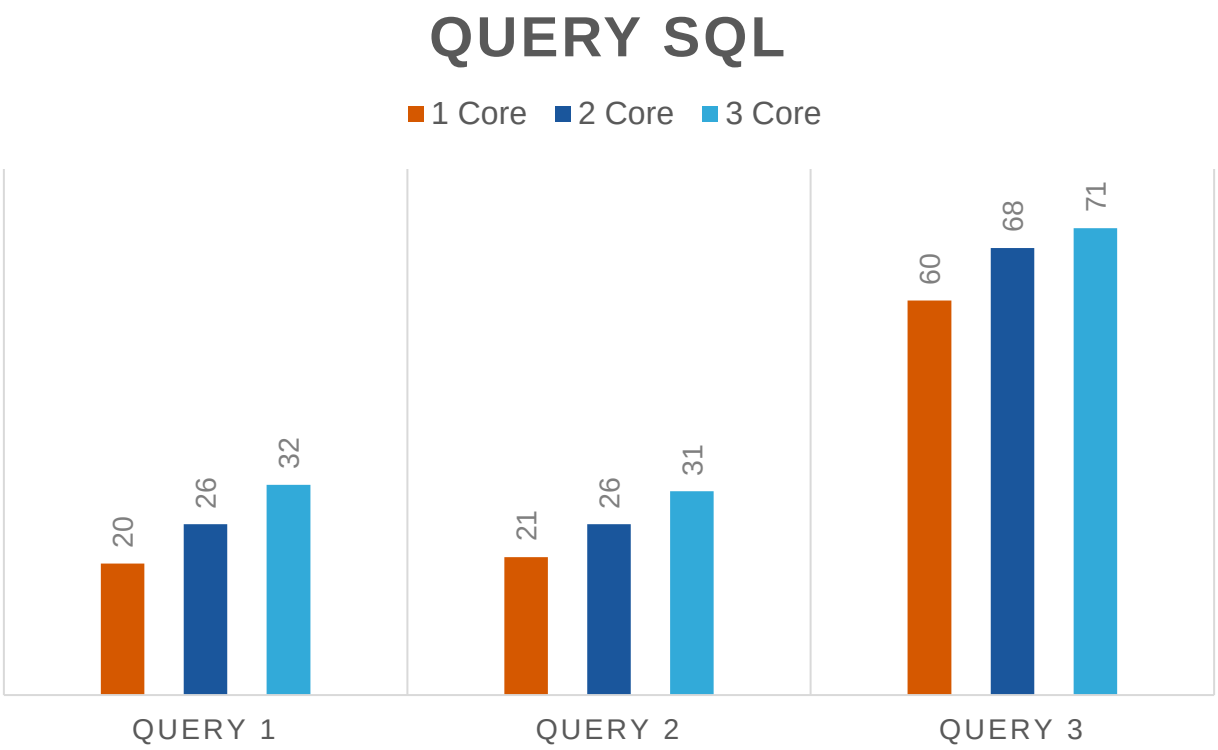


	_id ObjectId	date String	vault_id Int32	failure_count Int32
1	ObjectId('666422d1daa3b8…	"04-04-2023"	1066	2
2	ObjectId('666422d1daa3b8…	"04-04-2023"	1113	3
3	ObjectId('666422d1daa3b8…	"05-04-2023"	1010	2
4	ObjectId('666422d1daa3b8…	"05-04-2023"	1014	2
5	ObjectId('666422d1daa3b8…	"05-04-2023"	1032	3
6	ObjectId('666422d1daa3b8…	"05-04-2023"	1041	2
7	ObjectId('666422d1daa3b8…	"05-04-2023"	1044	2
8	ObjectId('666422d1daa3b8…	"05-04-2023"	1093	3
9	ObjectId('666422d1daa3b8…	"07-04-2023"	1033	2
10	ObjectId('666422d1daa3b8…	"07-04-2023"	1133	2
11	ObjectId('666422d1daa3b8…	"08-04-2023"	1124	2
12	ObjectId('666422d1daa3b8…	"08-04-2023"	1400	2
13	ObjectId('666422d1daa3b8…	"11-04-2023"	1036	2
14	ObjectId('666422d1daa3b8…	"11-04-2023"	1040	3
15	ObjectId('666422d1daa3b8…	"11-04-2023"	1090	3
16	ObjectId('666422d1daa3b8…	"11-04-2023"	1092	2
17	ObjectId('666422d1daa3b8…	"11-04-2023"	1093	2
18	ObjectId('666422d1daa3b8…	"12-04-2023"	1096	2
19	ObjectId('666422d1daa3b8…	"12-04-2023"	1113	3
20	ObjectId('666422d1daa3b8…	"14-04-2023"	1120	4

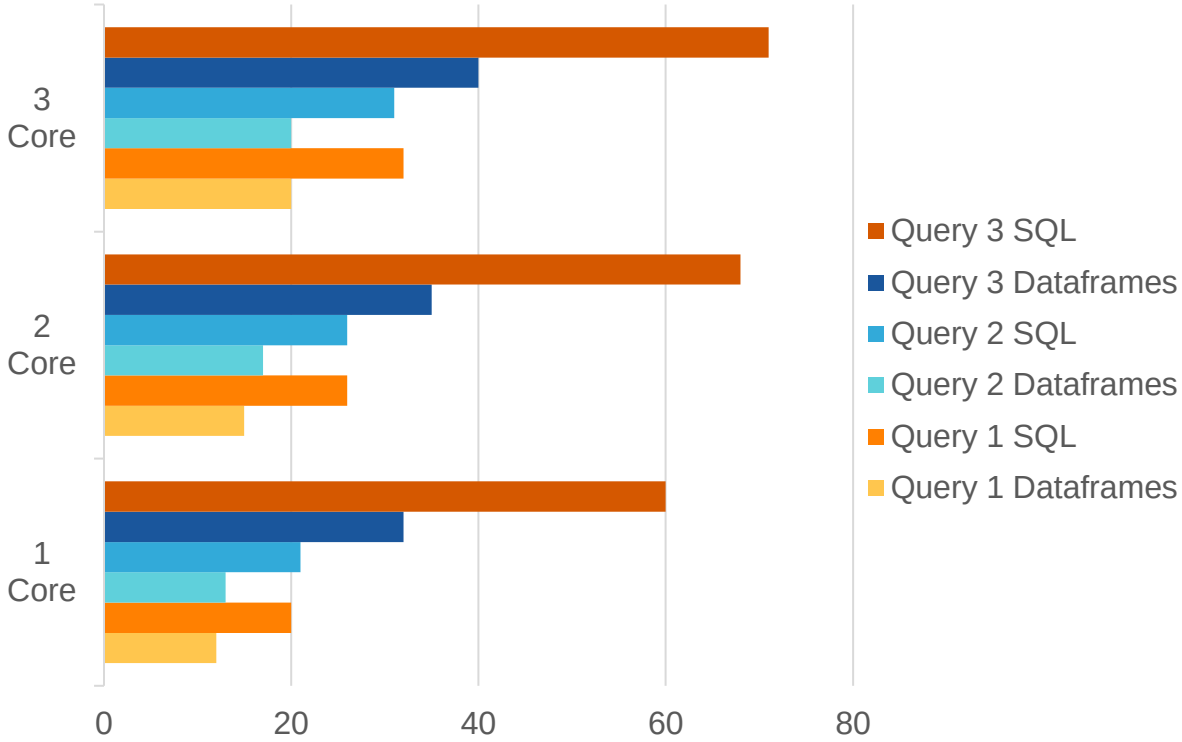
Analisi


QUERY DATAFRAMES
1 Core
2 Core
3 Core
12
15
20
13
17
20
32
35
40
QUERY 1
QUERY 2
QUERY 3

Analisi

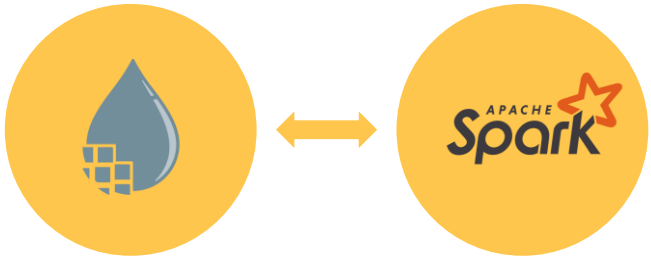

QUERY SQL
1 Core
2 Core
3 Core
20
26
32
21
26
31
60
68
71
QUERY 1
QUERY 2
QUERY 3

Analisi


3 Core
2 Core
1 Core
0
20
40
60
80
Query 3 SQL
Query 3 Dataframes
Query 2 SQL
Query 2 Dataframes
Query 1 SQL
Query 1 Dataframes

Conclusioni

Miglioramenti



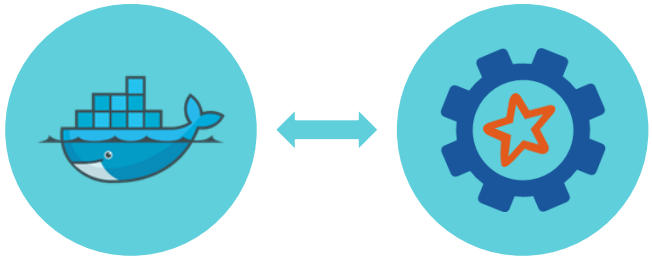
Nifi

Automatizzazione e persistenza di Nifi

RDD

Confronto delle prestazioni con RDD ed altri linguaggi di programmazione

Difficoltà



Architettura

Deployment iniziale dell'architettura e comunicazione tra i vari framework

Query

Svolgimento della Query 3 e interpretazione dei DAG

Grazie per l'attenzione!

https://github.com/Pi3raldoSanturro/SABD-Project1.git