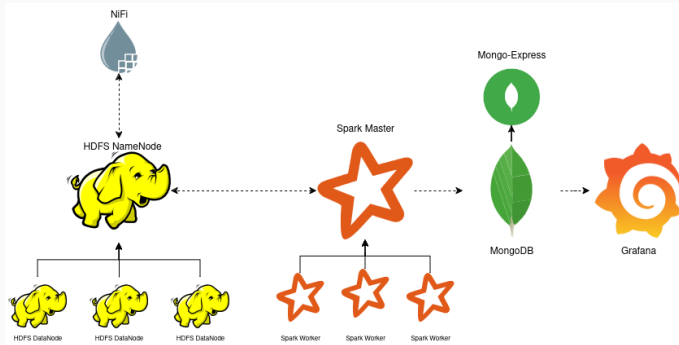


Analisi di dati di monitoraggio con Apache Spark

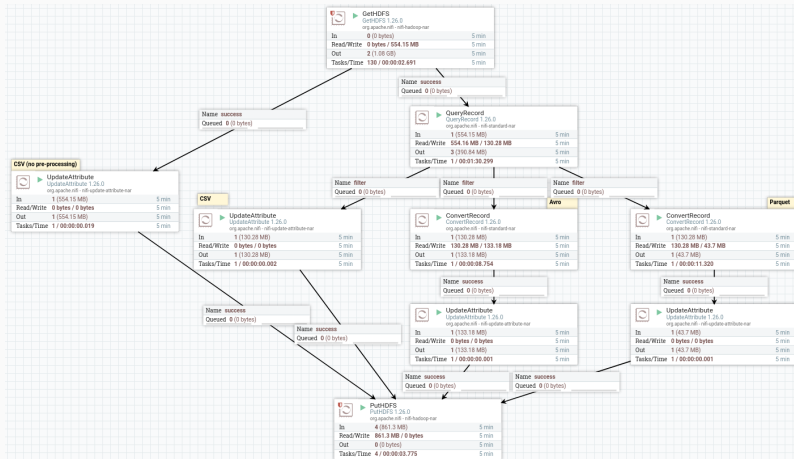
Sistemi e Architetture per Big Data - Progetto 1

Alessandro Lioi, 0333693



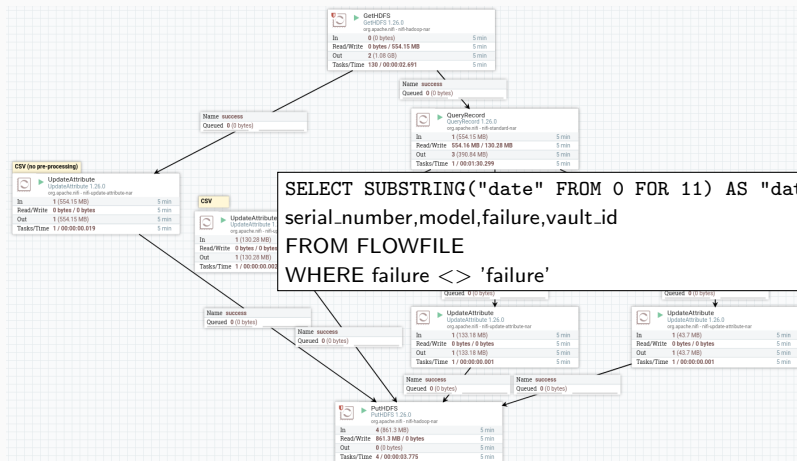
Componenti:

- HDFS: file system distribuito
- NiFi: pre-processing e data ingestion
- Spark: batch processing
- MongoDB: data store NoSQL
- Grafana: visualizzazione



Processor usati

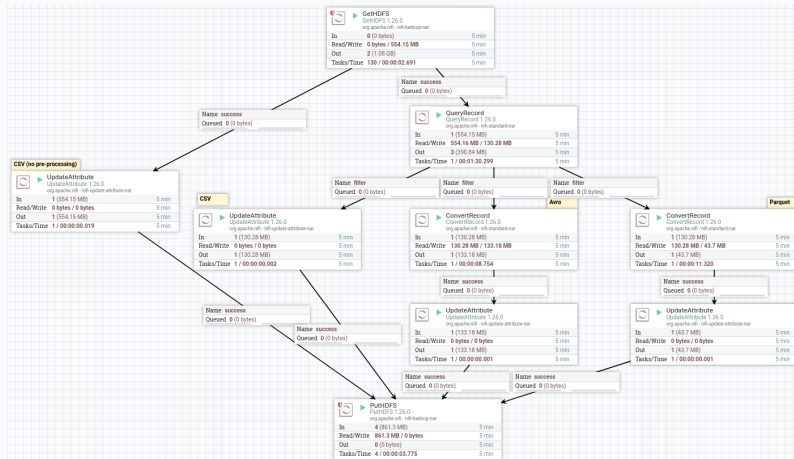
- **GetHDFS**: legge file da HDFS
- **QueryRecord**: seleziona le colonne di interesse
- **ConvertRecord**: converte dati da CSV in:
 - Apache Avro
 - Apache Parquet
 - CSV
- **UpdateAttribute**: cambia nome file output
- **PutHDFS**: salva su HDFS



Processor usati

- **GetHDFS**: legge file da HDFS
- **QueryRecord**: seleziona le colonne di interesse
- **ConvertRecord**: converte dati da CSV in:
 - Apache Avro
 - Apache Parquet
 - CSV
- **UpdateAttribute**: cambia nome file output
- **PutHDFS**: salva su HDFS

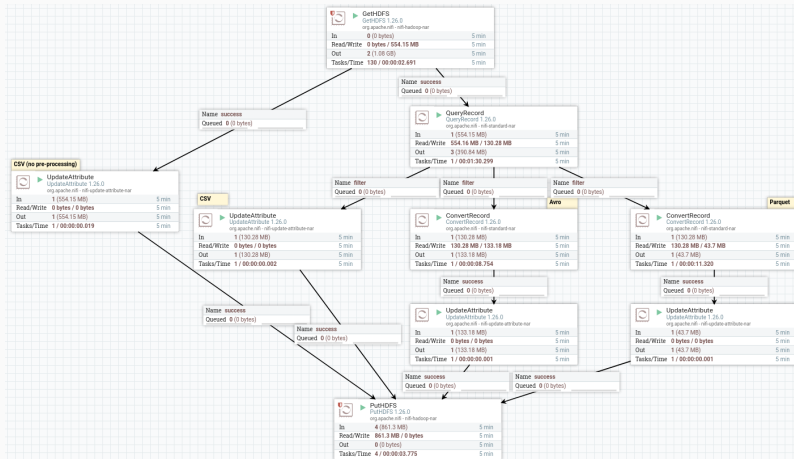
Data Ingestion e Pre-Processing



Processor usati

- **GetHDFS:** legge file da HDFS
- **QueryRecord:** seleziona le colonne di interesse
- **ConvertRecord:** converte dati da CSV in:
 - Apache Avro
 - Apache Parquet
 - CSV
- **UpdateAttribute:** cambia nome file output
- **PutHDFS:** salva su HDFS

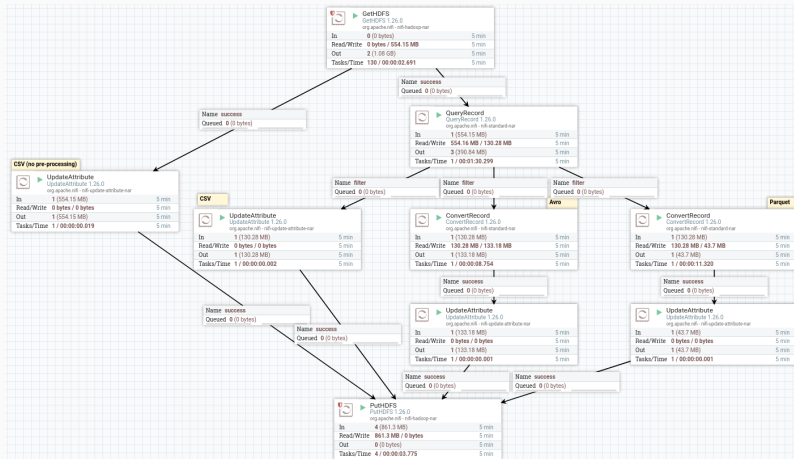
Data Ingestion e Pre-Processing



Processor usati

- **GetHDFS:** legge file da HDFS
- **QueryRecord:** seleziona le colonne di interesse
- **ConvertRecord:** converte dati da CSV in:
 - Apache Avro
 - Apache Parquet
 - CSV
- **UpdateAttribute:** cambia nome file output
- **PutHDFS:** salva su HDFS

Data Ingestion e Pre-Processing



Processor usati

- **GetHDFS**: legge file da HDFS
- **QueryRecord**: seleziona le colonne di interesse
- **ConvertRecord**: converte dati da CSV in:
 - Apache Avro
 - Apache Parquet
 - CSV
- **UpdateAttribute**: cambia nome file output
- **PutHDFS**: salva su HDFS

Query 1 - RDD

Tupla Originale:

date	serial_number	model	failure	vault_id
------	---------------	-------	---------	----------

Tupla Risultante

key		value
date	vault_id	failure

```
rdd.map(lambda x: ((x[0], x[4]), x[3]))  
    .reduceByKey(add)  
    .filter(lambda x: x[1] in {2, 3, 4})  
    .map(lambda x: (x[0][0], x[0][1], x[1]))  
    .sortBy(lambda x: (-x[2], x[0], x[1]))
```


Query 1 - RDD

Tupla Originale:

date	serial_number	model	failure	vault_id
------	---------------	-------	---------	----------

Tupla Risultante

key		value
date	vault_id	failures_count

```
rdd.map(lambda x: ((x[0], x[4]), x[3]))  
    .reduceByKey(add)  
    .filter(lambda x: x[1] in {2, 3, 4})  
    .map(lambda x: (x[0][0], x[0][1], x[1]))  
    .sortBy(lambda x: (-x[2], x[0], x[1]))
```

Query 1 - RDD

Tupla Originale:

date	serial_number	model	failure	vault_id
------	---------------	-------	---------	----------

Tupla Risultante

key		value
date	vault_id	failures_count in 2,3,4

```
rdd.map(lambda x: ((x[0], x[4]), x[3]))  
    .reduceByKey(add)  
    .filter(lambda x: x[1] in {2, 3, 4})  
    .map(lambda x: (x[0][0], x[0][1], x[1]))  
    .sortBy(lambda x: (-x[2], x[0], x[1]))
```

Query 1 - RDD

Tupla Originale:

date	serial_number	model	failure	vault_id
------	---------------	-------	---------	----------

Tupla Risultante

date	vault_id	failures_count in 2,3,4
------	----------	-------------------------

```
rdd.map(lambda x: ((x[0], x[4]), x[3]))  
    .reduceByKey(add)  
    .filter(lambda x: x[1] in {2, 3, 4})  
    .map(lambda x: (x[0][0], x[0][1], x[1]))  
    .sortBy(lambda x: (-x[2], x[0], x[1]))
```

Query 1 - RDD

Tupla Originale:

date	serial_number	model	failure	vault_id
------	---------------	-------	---------	----------

Tupla Risultante

date ↑	vault_id ↑	failures_count in 2,3,4 ↓
--------	------------	---------------------------

```
rdd.map(lambda x: ((x[0], x[4]), x[3]))  
    .reduceByKey(add)  
    .filter(lambda x: x[1] in {2, 3, 4})  
    .map(lambda x: (x[0][0], x[0][1], x[1]))  
    .sortBy(lambda x: (-x[2], x[0], x[1]))
```

In maniera analoga è implementata la query con i DataFrame

```
df.drop("serial_number", "model")  
rdd.map(lambda x: ((x[0], x[4]), x[3]))  
    .reduceByKey(add)  
    .filter(lambda x: x[1] in {2, 3, 4})  
    .map(lambda x: (x[0][0], x[0][1], x[1]))  
    .sortBy(lambda x: (-x[2], x[0], x[1]))  
df.groupBy("date", "vault_id")  
  .agg(sum("failure").alias("count"))  
df.filter(df["count"].isin(4, 3, 2))  
  .orderBy(  
    ["count", "date", "vault_id"],  
    ascending=[False, True, True],  
  )
```

Query 2 Ranking 1 - RDD

Tupla Originale:

date	serial_number	model	failure	vault_id
------	---------------	-------	---------	----------

Tupla Risultante

key	value
model	failure

```
ranking_1 = rdd.map(lambda x: (x[2], x[3]))  
                .reduceByKey(add)  
                .sortBy(lambda x: (-x[1], x[0]))  
df_ranking_1 = (  
    ranking_1.toDF(["model", "failures_count"])  
    .limit(10)  
)
```

Query 2 Ranking 1 - RDD

Tupla Originale:

date	serial_number	model	failure	vault_id
------	---------------	-------	---------	----------

Tupla Risultante

key	value
model	failures_count

```
ranking_1 = rdd.map(lambda x: (x[2], x[3]))  
                .reduceByKey(add)  
                .sortBy(lambda x: (-x[1], x[0]))  
df_ranking_1 = (  
    ranking_1.toDF(["model", "failures_count"])  
    .limit(10)  
)
```

Query 2 Ranking 1 - RDD

Tupla Originale:

date	serial_number	model	failure	vault_id
------	---------------	-------	---------	----------

Tupla Risultante

key	value
model ↑	failures_count ↓

```
ranking_1 = rdd.map(lambda x: (x[2], x[3]))  
                .reduceByKey(add)  
                .sortBy(lambda x: (-x[1], x[0]))  
df_ranking_1 = (  
    ranking_1.toDF(["model", "failures_count"])  
    .limit(10)  
)
```


- si trasforma in DataFrame
- si limitano a 10 risultati

```
ranking_1 = rdd.map(lambda x: (x[2], x[3]))  
                .reduceByKey(add)  
                .sortBy(lambda x: (-x[1], x[0]))  
df_ranking_1 = (  
    ranking_1.toDF(["model", "failures_count"])  
    .limit(10)  
)
```

In maniera analoga è implementata la query con i DataFrame

```
ranking_1 = rdd.map(lambda x: (x[2], x[3]))  
                .reduceByKey(add)  
                .sortBy(lambda x: (-x[1], x[0]))  
ranking_1.toDF(["model", "failures_count"])  
            .limit(10)
```

```
df.drop("date", "serial_number", "vault_id")  
   .groupBy("model")  
   .agg(sum("failure").alias("failures_count"))  
   .orderBy(["failures_count", "model"], ascending=False)  
   .limit(10)
```

The diagram illustrates the mapping of RDD operations to DataFrame operations. Red arrows connect the RDD code on the left to the DataFrame code on the right:

- `ranking_1 = rdd.map(lambda x: (x[2], x[3]))` maps to `df.drop("date", "serial_number", "vault_id")`
- `.reduceByKey(add)` maps to `.groupBy("model")`
- `.sortBy(lambda x: (-x[1], x[0]))` maps to `.agg(sum("failure").alias("failures_count"))`
- `ranking_1.toDF(["model", "failures_count"])` maps to `.orderBy(["failures_count", "model"], ascending=False)`
- `.limit(10)` maps to `.limit(10)`

Query 2 Ranking 2 - RDD

Tupla Originale:

date	serial_number	model	failure	vault_id
------	---------------	-------	---------	----------

Vault Failures

key	value
vault_id	failure

```
vault_failures = rdd.map(lambda x: (x[4], x[3]))  
                    .reduceByKey(add)  
vault_models = rdd.filter(lambda x: x[3] > 0)  
                .map(lambda x: (x[4], x[2]))  
                .groupByKey()  
                .mapValues(set)  
vault_failures.join(vault_models)  
    .map(lambda x: (x[0], int(x[1][0]), ",".join(x[1][1])))  
    .sortBy(lambda x: (-x[1], x[0]))  
    .toDF(["vault_id", "failures_count", "list_of_models"])  
    .limit(10)
```

Query 2 Ranking 2 - RDD

Tupla Originale:

date	serial_number	model	failure	vault_id
------	---------------	-------	---------	----------

Vault Failures

key	value
vault_id	failures_count

```
vault_failures = rdd.map(lambda x: (x[4], x[3]))  
                        .reduceByKey(add)  
vault_models = rdd.filter(lambda x: x[3] > 0)  
                  .map(lambda x: (x[4], x[2]))  
                  .groupByKey()  
                  .mapValues(set)  
vault_failures.join(vault_models)  
                .map(lambda x: (x[0], int(x[1][0]), ",".join(x[1][1])))  
                .sortBy(lambda x: (-x[1], x[0]))  
                .toDF(["vault_id", "failures_count", "list_of_models"])  
                .limit(10)
```

Query 2 Ranking 2 - RDD

Vault Failures

key	value
vault_id	failures_count

Vault Models

date	s_num	model	failure > 0	vault_id
------	-------	-------	-------------	----------

```
vault_failures = rdd.map(lambda x: (x[4], x[3]))  
                        .reduceByKey(add)  
vault_models = rdd.filter(lambda x: x[3] > 0)  
                        .map(lambda x: (x[4], x[2]))  
                        .groupByKey()  
                        .mapValues(set)  
vault_failures.join(vault_models)  
                .map(lambda x: (x[0], int(x[1][0]), ",".join(x[1][1])))  
                .sortBy(lambda x: (-x[1], x[0]))  
                .toDF(["vault_id", "failures_count", "list_of_models"])  
                .limit(10)
```

Query 2 Ranking 2 - RDD

Vault Failures

key	value
vault_id	failures_count

Vault Models

key	value
vault_id	set_of_model

```
vault_failures = rdd.map(lambda x: (x[4], x[3]))  
                        .reduceByKey(add)  
vault_models = rdd.filter(lambda x: x[3] > 0)  
                        .map(lambda x: (x[4], x[2]))  
                        .groupByKey()  
                        .mapValues(set)  
vault_failures.join(vault_models)  
    .map(lambda x: (x[0], int(x[1][0]), ",".join(x[1][1])))  
    .sortBy(lambda x: (-x[1], x[0]))  
    .toDF(["vault_id", "failures_count", "list_of_models"])  
    .limit(10)
```

Ranking 2

vault_id	failures_count	set_of_models
----------	----------------	---------------

```
vault_failures = rdd.map(lambda x: (x[4], x[3]))  
                      .reduceByKey(add)  
vault_models = rdd.filter(lambda x: x[3] > 0)  
                  .map(lambda x: (x[4], x[2]))  
                  .groupByKey()  
                  .mapValues(set)  
vault_failures.join(vault_models)  
    .map(lambda x: (x[0], int(x[1][0]), ",".join(x[1][1])))  
    .sortBy(lambda x: (-x[1], x[0]))  
    .toDF(["vault_id", "failures_count", "list_of_models"])  
    .limit(10)
```

Ranking 2

vault_id ↑	failures_count ↓	list_of_models
------------	------------------	----------------

```
vault_failures = rdd.map(lambda x: (x[4], x[3]))  
                    .reduceByKey(add)  
vault_models = rdd.filter(lambda x: x[3] > 0)  
                .map(lambda x: (x[4], x[2]))  
                .groupByKey()  
                .mapValues(set)  
vault_failures.join(vault_models)  
                .map(lambda x: (x[0], int(x[1][0]), ",".join(x[1][1])))  
                .sortBy(lambda x: (-x[1], x[0]))  
                .toDF(["vault_id", "failures_count", "list_of_models"])  
                .limit(10)
```

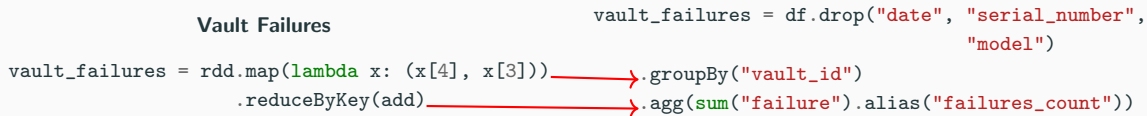

- si trasforma in DataFrame
- si limitano a 10 risultati

```
vault_failures = rdd.map(lambda x: (x[4], x[3]))  
                    .reduceByKey(add)  
vault_models = rdd.filter(lambda x: x[3] > 0)  
                .map(lambda x: (x[4], x[2]))  
                .groupByKey()  
                .mapValues(set)  
vault_failures.join(vault_models)  
    .map(lambda x: (x[0], int(x[1][0]), ",".join(x[1][1])))  
    .sortBy(lambda x: (-x[1], x[0]))  
    .toDF(["vault_id", "failures_count", "list_of_models"])  
    .limit(10)
```

In maniera analoga è implementata la query con i DataFrame

Vault Failures

```
vault_failures = df.drop("date", "serial_number",  
                          "model")  
  
vault_failures = rdd.map(lambda x: (x[4], x[3]))  
                    .groupBy("vault_id")  
                    .reduceByKey(add)  
                    .agg(sum("failure").alias("failures_count"))
```

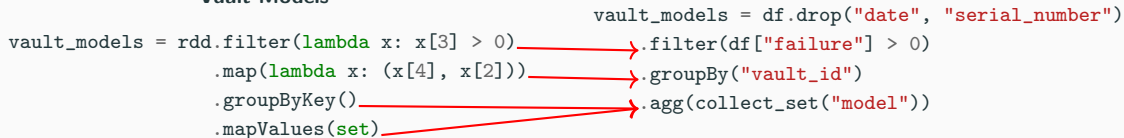


In maniera analoga è implementata la query con i DataFrame

Vault Models

```
vault_models = rdd.filter(lambda x: x[3] > 0)
                    .map(lambda x: (x[4], x[2]))
                    .groupByKey()
                    .mapValues(set)

vault_models = df.drop("date", "serial_number")
                  .filter(df["failure"] > 0)
                  .groupBy("vault_id")
                  .agg(collect_set("model"))
```



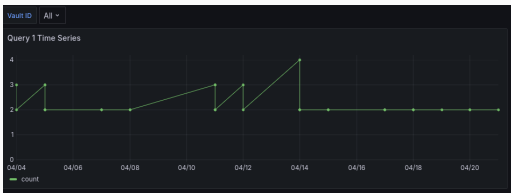
Query 2 Ranking 2 - DataFrame

In maniera analoga è implementata la query con i DataFrame

Ranking 2

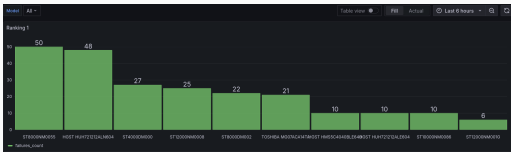
```
vault_failures.join(vault_models)
  .map(lambda x:
        (x[0], int(x[1][0]), ",".join(x[1][1]))
      )
  .sortBy(lambda x: (-x[1], x[0]))
  .toDF([...])
  .limit(10)
```

df_ranking_2 = vault_failures.join(
 vault_models,
 "vault_id",
 how="left")
 .withColumn(
 "list_of_models",
 concat_ws(",",
 vault_models["collect_set(model)"]
),
)
 .drop("collect_set(model)")
 .orderBy(["failures_count", "vault_id"], ascend
)
 .limit(10)



Query 1

- Time Series
- Filtro sul Vault ID



Query 2 Ranking 1

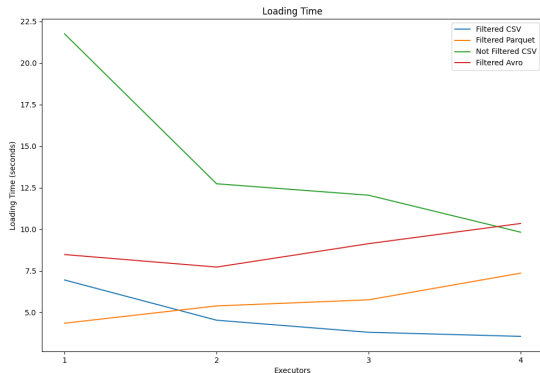
- Bar Chart
- Filtro sul Modello



Query 2 Ranking 2

- Bar Chart
- Filtro sul Modello e Vault ID

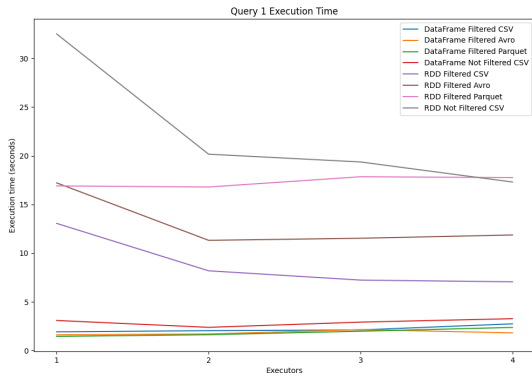
Performance - Tempi di Caricamento



- CSV filtrato più veloce all'aumentare degli executors
- Parquet formato con occupazione migliore

Formato	Peso
CSV Originale	581M
CSV Filtrato	137M
Apache Avro	140M
Apache Parquet	46M

Performance - Tempi di Esecuzione - Query 1



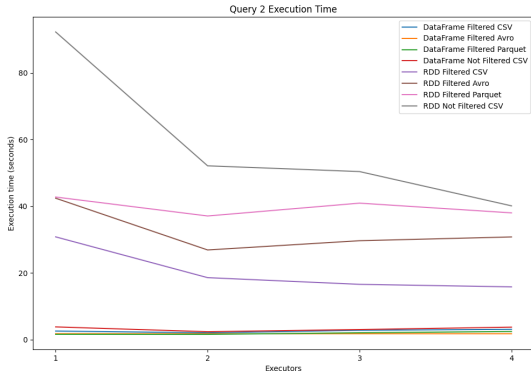
RDD

- CSV formato migliore
- Performance migliori all'aumentare degli executors

DataFrame

- Parquet formato migliore
- Prestazioni migliori

Performance - Tempi di Esecuzione - Query 2



RDD

- Situazione analoga alla Query 1

DataFrame

- Parquet e Avro formati migliori