Analisi di dati energetici con Apache Spark

Marco Lorenzini
Università di Tor Vergata
Roma, Italia
marcolorenzini23@gmail.com

***Abstract*—In questo lavoro descriviamo l'applicazione di Apache Spark per l'elaborazione e l'interrogazione di dati storici orari relativi alla Carbon Intensity e alla Carbon-Free Energy Percentage in Italia e in Svezia tra il 2021 e il 2024. I dataset, forniti da Electricity Maps, includono serie temporali dettagliate di energia prodotta e intensità emissiva. Descriviamo l'architettura Spark adottata, illustrando la pipeline di ingestione, pulizia e trasformazione dei dati, e mostriamole query effettuate, con un'analisi delle prestazioni in termini di tempo di esecuzione e scalabilità.**

I. Introduzione

Le query a cui andremo a rispondere sono le seguenti:

A. Query 1

Facendo riferimento al dataset dei valori energetici dell'Italia e della Svezia, aggregare i dati su base annua. Calcolare la media, il minimo ed il massimo di "Carbon intensity gCO2eq/kWh (direct)" e "Carbon-free energy percentage (CFE%)" per ciascun anno dal 2021 al 2024. Inoltre, considerando il valore medio di "Carbon intensity gCO2eq/kWh (direct)" e "Carbon-free energy percentage (CFE%)" aggregati su base annua, generare due grafici che consentano di confrontare visivamente l'andamento per Italia e Svezia.

B. Query 2

Considerando il solo dataset italiano, aggregare i dati sulla coppia (anno, mese), calcolando il valor medio di "Carbon intensity gCO2eq/kWh (direct)" e "Carbon-free energy percentage (CFE%)". Calcolare la classifica delle prime 5 coppie (anno, mese) ordinando per "Carbon intensity gCO2eq/kWh (direct)" decrescente, crescente e "Carbon-free energy percentage (CFE%)" decrescente, crescente. In totale sono attesi 20 valori. Inoltre, considerando il valor medio di "Carbon intensity gCO2eq/kWh (direct)" e "Carbon-free energy percentage (CFE%)" aggregati sulla coppia (anno, mese) per l'Italia, generare due grafici che consentano di valutare visivamente l'andamento delle due metriche.

C. Query 3

Facendo riferimento al dataset dei valori energetici dell'Italia e della Svezia, aggregare i dati di ciascun paese sulle 24 ore della giornata, calcolando il valor medio di "Carbon intensity gCO2eq/kWh (direct)" e "Carbon-free energy percentage (CFE%)". Calcolare il minimo, 25-esimo, 50-esimo, 75-esimo percentile e massimo del valor medio di "Carbon intensity gCO2eq/kWh (direct)" e "Carbonfree energy percentage (CFE%)". Inoltre, considerando il valor medio di "Carbon intensity gCO2eq/kWh (direct)" e "Carbon-free energy percentage (CFE%)" aggregati sulle 24 fasce orarie giornaliere, generare due grafici che consentano di confrontare visivamente l'andamento per Italia e Svezia.

II. Architettura del sistema

Di seguito riportiamo l'architettura generale del sistema:

A. Apache NiFi

Apache NiFi si occupa dell'ingestione e della trasformazione dei dati nella fase iniziale, e dell'esportazione dei risultati finali delle query da HDFS a Redis.

B. HDFS

HDFS funge da data lake: ospita sia i file in formato Parquet e CSV prodotti da NiFi sia i risultati finali delle elaborazioni Spark in formato CSV.

C. Apache Spark

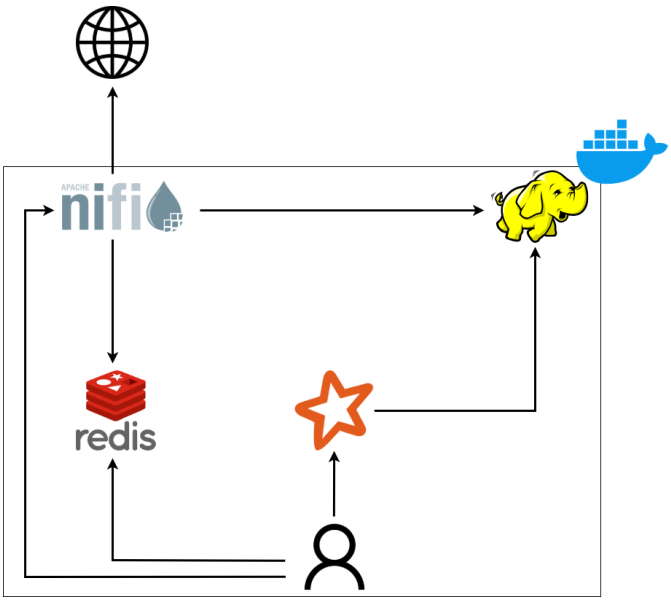
Apache Spark accede ai dati Parquet e CSV su HDFS per eseguire le query. Grazie alla natura distribuita del cluster, formata in questo specifico caso da un nodo Master, uno o più Worker e un History Server, permette il processamento di grandi volumi di dati con elevate prestazioni.

D. Redis

Redis viene utilizzato come store a bassa latenza per i risultati finali delle query.

E. Docker

Tutti i componenti (NiFi, HDFS, Spark, Redis) sono containerizzati tramite Docker, semplificando il deployment, l'isolamento e la scalabilità dei servizi.
Vediamo uno schema riassuntivo sull'architettura:



III. Applicazione del sistema

A. Tecnologie e librerie

L'esecuzione delle query avviene tramite un'applicazione Python che utilizza il cluster Apache Spark descritto in precedenza. Si sfrutta la libreria `pyspark`, fornita dal framework, per interfacciarsi con il cluster.

B. Ingestione dei dati e trasformazione (Data Ingestion and Transformation)

Questa fase viene realizzata tramite NiFi.

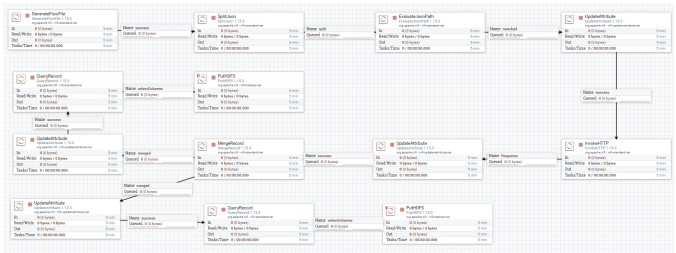


Fig. 1. Data Ingestion e Transformation con NiFi

L'obiettivo del Process Group è effettuare il fetch dei dati dal sito di Electricity Maps ed eseguire il preprocessing. Si parte dal processore `GenerateFlowFile`, che produce in output un file JSON del tipo:

```
[
  {"country": "IT", "year": "2021"},
  {"country": "SE", "year": "2021"},
  ...
]
```

I processori `SplitJson` e `EvaluateJsonPath` estraggono quindi i valori di `country` e `year`, in modo da costruire i percorsi per il download dei dataset. Successivamente, `UpdateAttribute` crea l'URL di download con la seguente sintassi:

```
https://data.electricitymaps.com/2025-04-03
/${country}_${year}_hourly.csv
```

Questo URL viene passato al processore `InvokeHTTP`, che scarica i file CSV corrispondenti. A valle del download, un ulteriore `UpdateAttribute` rinomina i file in base alla `country` e all'`year`, per semplicità di gestione. I file rinominati vengono quindi inviati al processore `MergeRecord`, che li unisce tramite la proprietà `Correlation Attribute Name` impostata a `country`. Dopo la fusione, 2 diversi `UpdateAttribute` rinomina nuovamente il file, basandosi esclusivamente sul valore di `country` e inserendo il corretto formato (.csv e .parquet). Infine, due processori `QueryRecord` si occupano di eliminare le colonne non necessarie e di rinominare quelle di interesse. Sono necessari due `QueryRecord` distinti perché il formato di output è diverso: uno genera Parquet e l'altro CSV. I file risultanti vengono quindi inviati a due processori `PutHDFS`, che si occupano di caricarli in HDFS in cartelle separate per Parquet e CSV.

IV. Definizione delle query

Per eseguire le tre query, il codice viene prima copiato all'interno del container master tramite un volume Docker temporaneo. Successivamente, lo script viene sottoposto a Spark, e avviato sul cluster per l'elaborazione. Notiamo inoltre che le query eseguite su RDD utilizzano file in formato CSV, mentre le query con DataFrame sfruttano sia CSV sia Parquet.

A. Query 1

1) RDD:

- Applichiamo innanzitutto tramite Map sull'RDD una funzione che ci permette di estrarre l'anno dal timestamp, e ci ritorna un output mappato come: `((Country, Year), (CarbonDirect, CFEpercent))`
- Definiamo la funzione sequenziale `seq_op` che, per ciascun valore `(carbon, cfe)`, mantiene in un array:
  - somma, minimo e massimo di `carbon`,
  - contatore dei record,
  - somma, minimo e massimo di `cfe`.
- Definiamo la funzione di combinazione `comb_op` per unire due stati parziali, sommando somme e contatori e confrontando minimi e massimi di `carbon` e `cfe`.
- Utilizziamo `aggregateByKey(zero, seq_op, comb_op)` per eseguire in un unico passaggio lo shuffle e calcolare, per ogni chiave `(Country, Year)`, la struttura `[sumC, minC, maxC, count, sumF, minF, maxF]`.
- Applichiamo `map`, dopo avert ordinato per chiave, per trasformare lo stato aggregato in tuple: `(Country, Year, avg_carbon, min_carbon, max_carbon, avg_cfe,`

`min_cfe, max_cfe)`, calcolando le medie come `somma/contatore`.

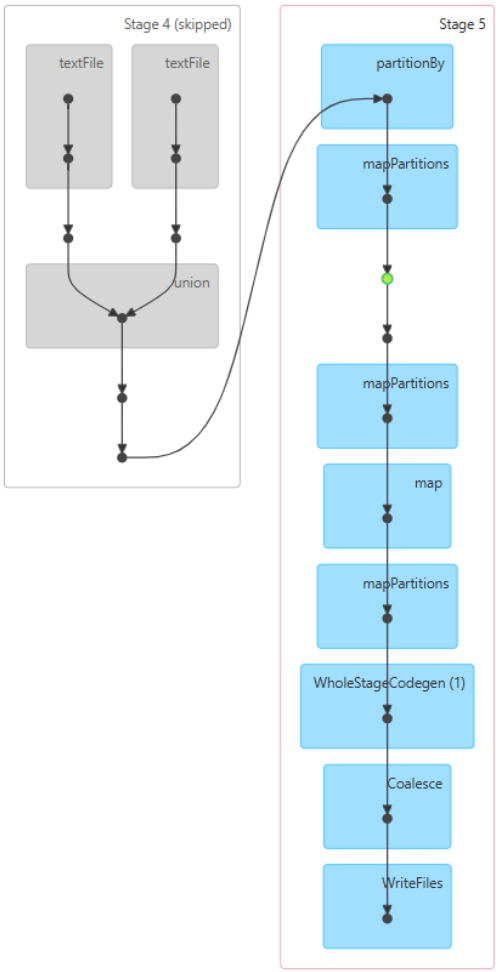


Fig. 2. Query 1 RDD DAG

2) DataFrame: Effettuiamo innanzitutto un'aggregazione annuale sui dati raggruppando per Country e anno, calcolando per ciascun gruppo le seguenti metriche:

- Intensità di carbonio diretta: media, valore minimo e valore massimo.
- Percentuale di energia carbon-free: media, valore minimo e valore massimo

I risultati vengono infine ordinati per paese e anno, in modo da ottenere una tabella cronologica per ciascuna nazione.

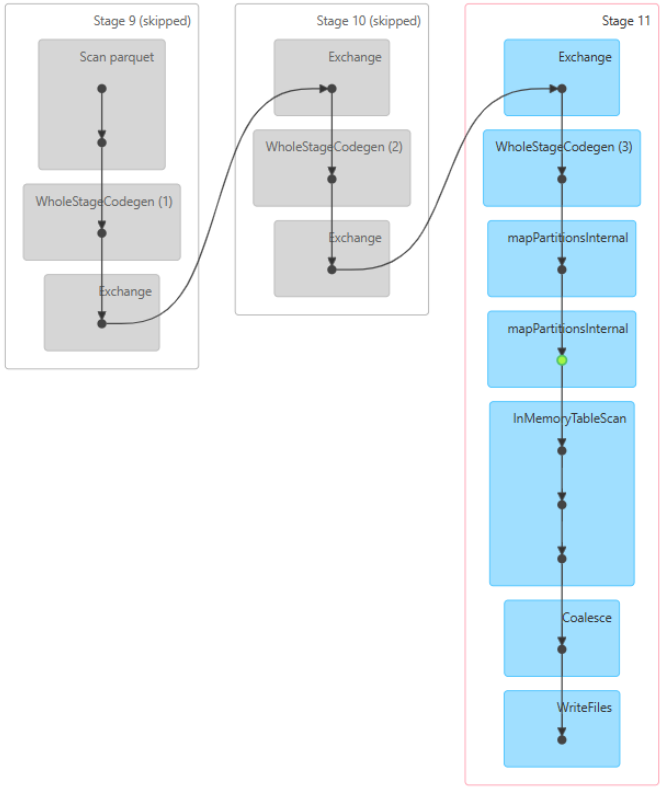


Fig. 3. Query 1 DF DAG

3) SQL: I passi sono:

- Creazione di una view temporanea `energy_data` a partire dal DataFrame caricato.
- Definizione ed esecuzione di una query SQL che:
  - Seleziona le colonne `Country` e `record_year`;
  - Calcola per ciascun gruppo (`Country`, `record_year`):
    - Intensità di carbonio diretta: `AVG(CarbonDirect)`, `MIN(CarbonDirect)`, `MAX(CarbonDirect)`;
    - Percentuale di energia carbon-free: `AVG(CFEpercent)`, `MIN(CFEpercent)`, `MAX(CFEpercent)`.
  - Raggruppa i risultati con `GROUP BY Country, record_year`;
  - Ordina cronologicamente per Paese e anno con `ORDER BY Country, record_year`.

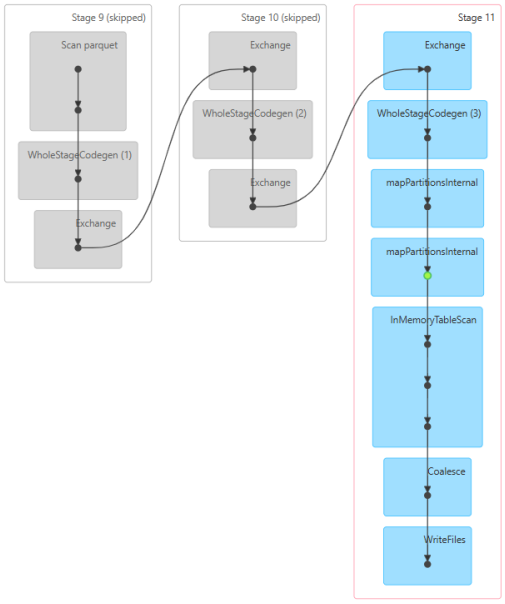
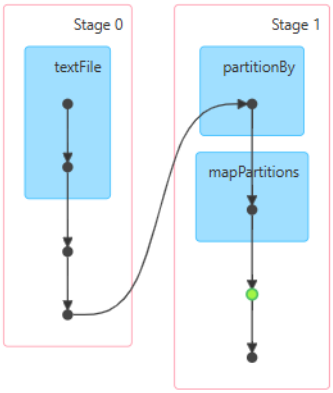


Fig. 4. Query 1 SQL DAG

B. *Query 2*

1) RDD:

- Applichiamo innanzitutto tramite Map sull'RDD una funzione che ci permette di estrarre l'anno dal timestamp, e ci ritorna un output mappato come: `((Year, Month), (CarbonDirect, CFEpercent,1 ))`
- Chiamiamo reduceByKey sull'RDD calcolando in questo modo le somme dei valori e il contatore.
- Adesso calcoliamo le medie dividendo somma/contatore e applichiamo `.cache()` per riutilizzare i dati.
- Estraiamo i primi 5 e gli ultimi 5 valori per intensità carbonica e per CFE% tramite `takeOrdered(5)` e `top(5)`.



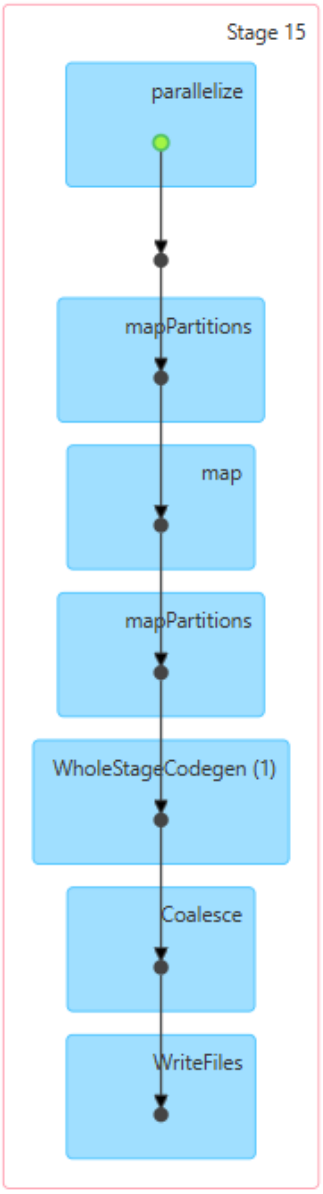


Fig. 5. Query 2 RDD

2) DataFrame: In questa query, dovendo trattare solamente l'Italia, carichiamo i dati da HDFS selezionando solo le righe con Country = 'IT'.
A questo punto raggruppiamo i dati su anno e mese, e calcoliamo, per ciascuna coppia, le medie di:

- Intensità di CO_2 diretta
- Percentuale di energia carbon-free

Successivamente applichiamo `.cache()` al DataFrame aggregato e lanciamo un'azione `.count()` per materializzare i risultati in memoria, in modo da accelerare le successive ordinazioni.
Procediamo con l'estrazione dei 5 valori più alti/bassi per entrambe le metriche sfruttando OrderBy.

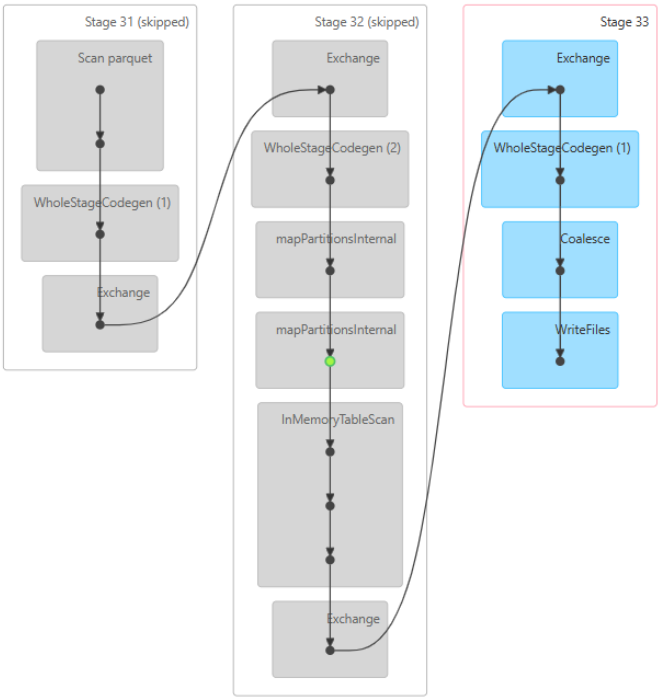


Fig. 6. Query 2 DF DAG

Fig. 7. Query 2 SQL DAG

3) SQL: Carichiamo i dati da HDFS filtrando su `Country='IT'` (nel caso Parquet, nel caso CSV invece carichiamo direttamente il file IT) e creiamo la view temporanea `energy_it`.
Definiamo una CTE `agg` che raggruppa per `record_year` e `record_month` e calcola le medie di:

- Intensità di CO_2 diretta (`AVG(CarbonDirect)`)
- Percentuale di energia carbon-free (`AVG(CFEpercent)`)

Applichiamo `.cache()` sul risultato di `spark.sql(sql)` e lanciamo `.count()` per materializzare in memoria.
Infine, per ciascuna metrica estraiamo i 5 record con valori più alti e più bassi utilizzando quattro query SQL con `ORDER BY ... LIMIT 5`.

C. Query 3

1) RDD:

- Estraiamo l'ora e mappiamo i record in una coppia `((Country, Hour), (valore_carbonDirect, valore_CFE, 1))` per preparare l'aggregazione oraria per ciascun paese.
- Chiamiamo una reduceByKey per la somma dei valori e del contatore.
- Dal primo RDD aggregato calcoliamo le medie orarie dividendo somma per contatore e riproiettiamo per paese: `(Country, (media_carbon_oraria, media_CFE_oraria))`.
- Applichiamo adesso groupByKey() su Country: così da raccogliere tutte le coppie (avg_carbon, avg_cfe) per lo stesso Paese in una lista.
- Usiamo mapValues per calcolare, per ciascun Paese, due serie di statistiche:
  - Dalla lista di tutti i valori avg_carbon, estraiamo solo la componente avg_carbon e chiamiamo calculate_stats su quell'elenco.
  - Dalla stessa lista, estraiamo i valori avg_cfe e chiamiamo di nuovo calculate_stats.

La funzione `calculate_stats` fornendosi anche di `calculate_percentile` si occupa del calcolo manuale con interpolazione lineare dei quantili e delle altre metriche necessarie.

- Trasformiamo i risultati tramite `flatMap` per ottenere righe separate per ciascuna metrica (carbon-intensity e cfe) per paese.

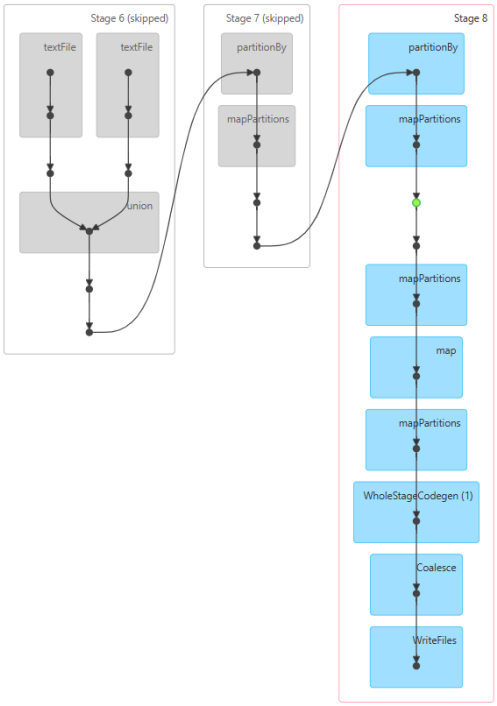


Fig. 8. RDD DAG

2) DataFrame: Questa query è stata realizzata, in versione DataFrame, in 2 modi diversi, utilizzando rispettivamente la funzione `percentile_approx` e la funzione `percentile` per confrontare la differenza di risultati e tempi tra le due.
Raggruppiamo i dati per country e ora, calcolando la media di:

- Intensità di CO_2 diretta
- Percentuale di energia carbon-free

Sul risultato intermedio, raggruppiamo per country e calcoliamo:

- min e max di ciascuna metrica
- Il calcolo dei quartili si differenzia tra le due versioni, nel caso dell'approssimata viene calcolato con valore dell'accuracy 100, questo valore controlla la precisione dell'approssimazione a scapito dell'uso di memoria. Nella versione esatta, viene utilizzata invece la funzione `percentile`, che calcola i percentili effettivi.

Il risultato finale combina in entrambi i casi le due serie di statistiche.

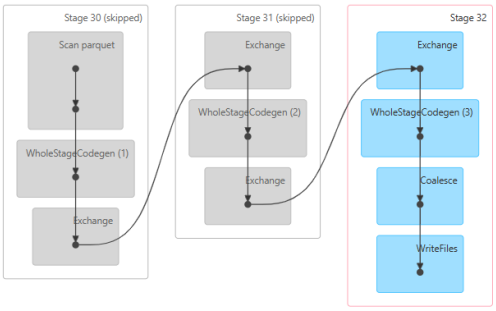


Fig. 9. Query 3 Exact/Approx DAG

3) SQL: Eseguiamo una singola query SQL con CTE sui dati di Italia e Svezia:

- **hourly**: raggruppa per `Country` e `record_hour`, calcolando
  - Intensità di CO_2 diretta (`AVG(CarbonDirect)`)
  - Percentuale di energia carbon-free (`AVG(CFEpercent)`)
- **stats_carbon** e **stats_cfe**: per ciascun `Country` calcolano sulle medie orarie
  - Minimo (`MIN`), 25° (`percentile(...,ARRAY(0.25))[0]`), 50° (`percentile(...,ARRAY(0.50))[0]`), 75° (`percentile(...,ARRAY(0.75))[0]`) e massimo (`MAX`)
- Uniamo i due set di statistiche con `UNION ALL` in un unico risultato.

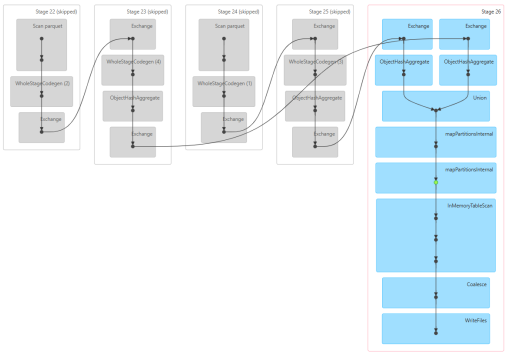


Fig. 10. Query 3 SQL

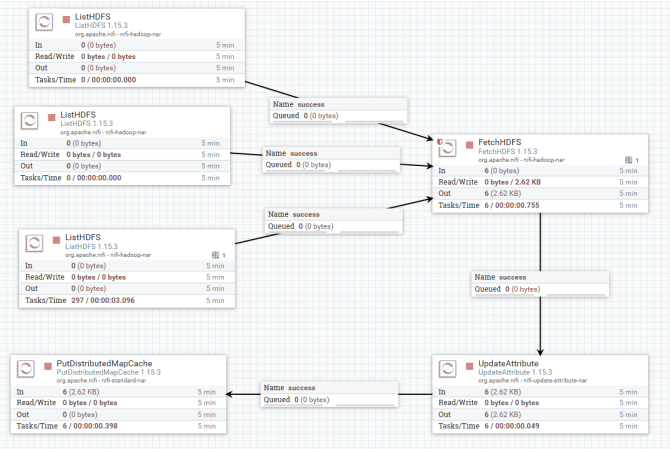
V. Scrittura dei risultati

I risultati di tutte le query vengono salvati, nel caso di DataFrame, in formato CSV su HDFS, utilizzando

```
.coalesce(1)
.write
.mode("overwrite")
.option("header", "true")
.csv(<HDFS_PATH>)
```

per produrre un singolo file con intestazione. Notiamo che nelle query con RDD estraiamo il DF e salviamo allo stesso

modo.
Successivamente, Apache NiFi estrae questi dati da HDFS e li carica in Redis per garantire accessi a bassa latenza. Il caricamento su Redis avviene tramite NiFi seguendo il seguente Process Group:



Il Process Group inizia con tre processori `ListHDFS`, che recuperano da HDFS l'elenco dei file presenti nelle cartelle relativi ai risultati delle query RDD (scelta fatta per semplificazione, essendo tutti i risultati uguali. Successivamente, `FetchHDFS` scarica ciascun file elencato. A questo punto, con `UpdateAttribute` impostiamo l'attributo `redis.key` pari al nome del file CSV. Infine, il processore `PutDistributedMapCache` invia i dati a Redis.

VI. Risultati Query

A. Query 1

Di seguito il grafico che rappresenta l'andamento per Italia e Svezia della media della carbon intensity.

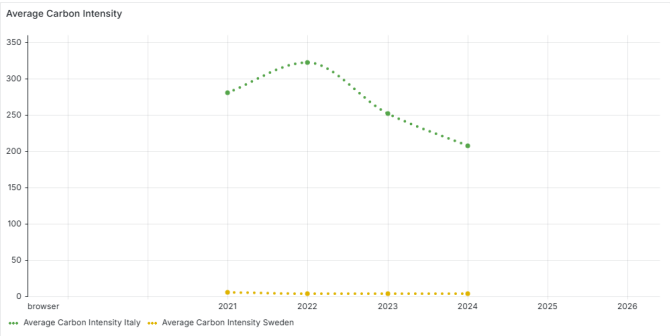


Fig. 11. Average Carbon Intensity - Year

Vediamo ora il grafico che rappresenta l'andamento per Italia e Svezia della media della Carbon-Free Energy in percentuale.

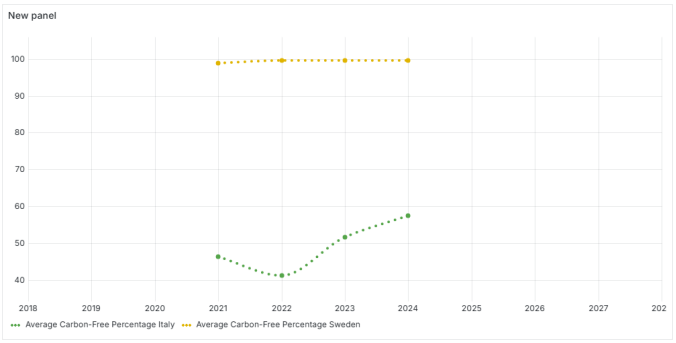


Fig. 12. Carbon-Free Energy (Percentual) - Year

B. Query 2

Di seguito il grafico che rappresenta l'andamento per l'Italia della media della Carbon Intensity con valori su anno e mese.

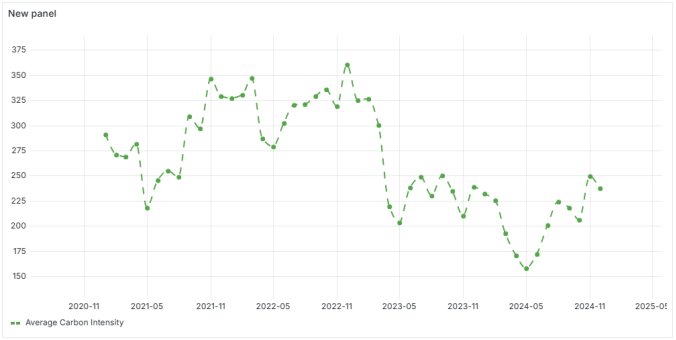


Fig. 13. Average Carbon Intensity - Year-Month

Vediamo ora il grafico che rappresenta l'andamento per l'Italia della media della Carbon-Free Energy in percentuale con valori su anno e mese.

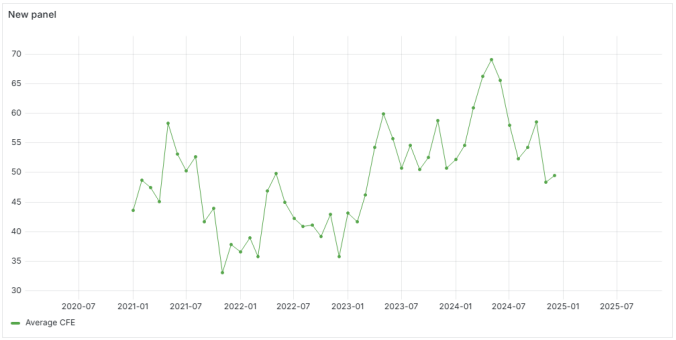


Fig. 14. Carbon-Free Energy (Percentual) - Year-Month

C. Query 3

Di seguito il grafico che rappresenta l'andamento per Italia e Svezia della media della Carbon Intensity sulle 24 ore.

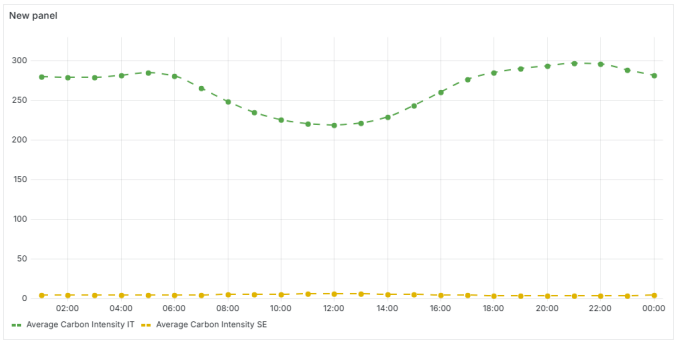


Fig. 15. Average Carbon Intensity - Daily

Vediamo ora il grafico che rappresenta l'andamento per Italia e Svezia della media della Carbon-Free Energy in percentuale con valori sulle 24 ore.

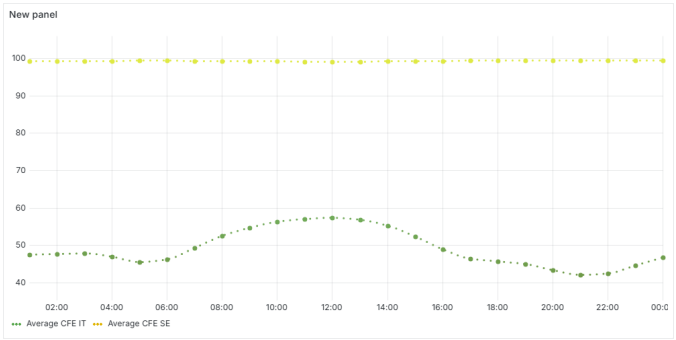


Fig. 16. Carbon-Free Energy (Percentual) - Daily

VII. Analisi Prestazioni

I tempi presi in considerazione comprendono la lettura del file e la relativa query, fino all'esecuzione di un azione count(), escludendo quindi i tempi di scrittura su HDFS. Inoltre, per uniformare tutti i risultati, ogni worker è stato avviato con le seguenti variabili d'ambiente:

```
SPARK_WORKER_MEMORY=1g
SPARK_WORKER_CORES=1
```

Riporto di seguito la tabella dove, per ogni Query, vengono esposti i tempi minimi, medi, massimi e la deviazione standard.

TABLE I
STATISTICHE AGGREGATE PER QUERY

Query Name	Mean	Std	Min	Max
QUERY_1_PARQUET	6.875443	0.824957	5.358830	9.189178
QUERY_2_PARQUET	7.211343	0.513084	6.355411	8.080244
QUERY_3_PARQUET	8.026729	1.129131	6.508313	10.508938
QUERY_3_EXACT_PARQUET	7.952566	1.051043	6.557607	9.675135
QUERY_1_SQL_PARQUET	6.934006	0.622228	6.040484	8.080536
QUERY_2_SQL_PARQUET	7.725713	0.593811	6.892687	8.770594
QUERY_3_SQL_PARQUET	7.941959	1.097861	6.347483	9.712888
QUERY_1_RDD	6.184908	0.955348	4.970206	7.889551
QUERY_2_RDD	5.992856	1.069987	4.734195	7.894956
QUERY_3_RDD	5.614439	0.962410	4.524328	7.327836
QUERY_1_CSV	6.648977	0.698307	5.458930	8.156664
QUERY_2_CSV	6.928772	0.472009	5.974462	7.791144
QUERY_3_CSV	7.878032	0.995227	6.622260	9.579778
QUERY_3_EXACT_CSV	7.929698	1.022130	6.428803	9.563535
QUERY_1_SQL_CSV	6.790257	0.638260	5.857209	8.211755
QUERY_2_SQL_CSV	7.376404	0.512777	6.690489	8.235862
QUERY_3_SQL_CSV	7.882526	1.063814	6.324144	10.016470

Vediamo i grafici dei tempi di esecuzione al variare del numero di worker per DF/SQL in versione CSV e Parquet.

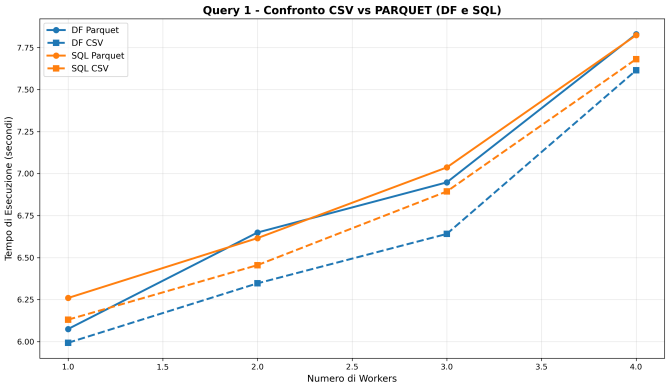


Fig. 17. Query 1

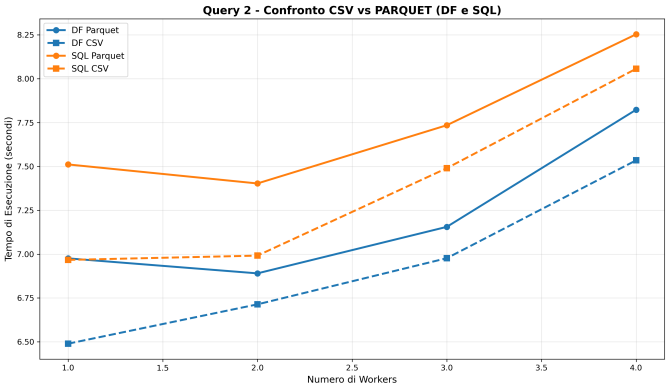


Fig. 18. Query 2

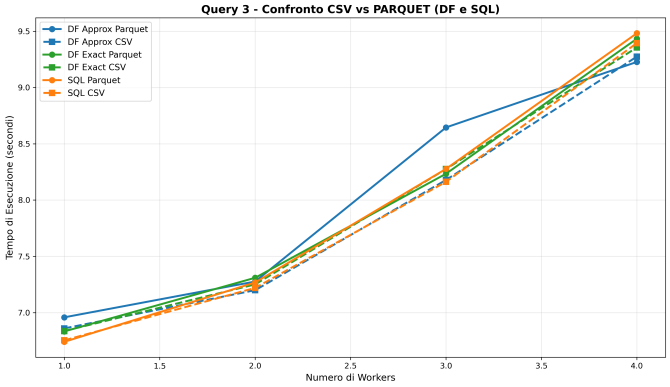


Fig. 19. Query 3

La prima osservazione riguarda il calo di prestazioni all'aumentare dei worker. Questo è un comportamento atteso quando si opera su un dataset di dimensioni contenute. L'overhead di Spark, che include lo scheduling dei task, la serializzazione dei dati, la comunicazione di rete tra i nodi e l'avvio degli executor, diventa il fattore dominante superando i benefici della parallelizzazione, portando a un tempo di esecuzione totale più lungo.
Notiamo che le Query eseguite con formato Parquet mostrano un tempo di esecuzione più elevato rispetto alle query eseguite con formato CSV, questo è atteso, infatti Parquet ha un overhead fisso per gestire i metadati che vista le esigue dimensione dei nostri file in input può diventare significativo rispetto ai vantaggi come la compressione e l'accesso colonnare.

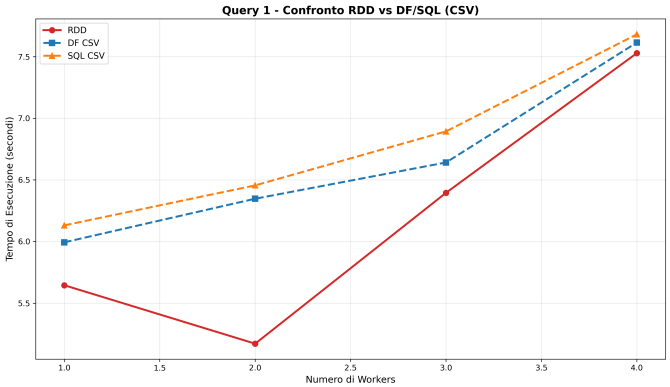


Fig. 20. Query 1

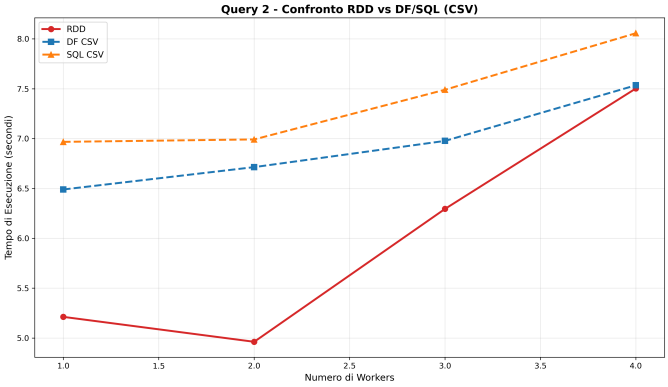


Fig. 21. Query 2

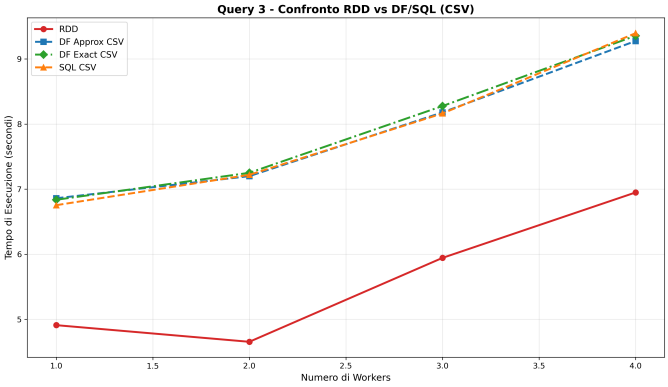


Fig. 22. Query 3

Come previsto, le query eseguite tramite API DataFrame (DF) e SQL mostrano curve di performance simili, con un leggero vantaggio per la versione DataFrame. Entrambe le interfacce, infatti, traducono il codice in un piano logico che viene poi ottimizzato dal medesimo motore, il Catalyst Optimizer. Poiché Catalyst produce piani di esecuzione fisici molto simili, le prestazioni risultano quasi identiche. Tuttavia, il codice SQL richiede un passaggio aggiuntivo di parsing per essere convertito in un albero sintattico, il che ne aumenta leggermente l'overhead.
L'implementazione RDD mostra invece un andamento differente, infatti risulta la più veloce, perché sfrutta operazioni primitive e non paga l'overhead del Catalyst Optimizer e la gestione della struttura astratta dei DataFrame ma scala però peggio rispetto a DF/SQL, fino ad avere un tempo di esecuzione maggiore/simile delle controparti SQL/DF. Questo può accadere perché, con un numero elevato di worker, il dataset (pur piccolo) viene suddiviso in troppi micro-task, aspetto invece meno pesante su DF, dove è presente l'Adaptive Query Execution (AQE). Se AQE è attivo (attivo di default nelle nuove versioni), Spark può "coalescere" partizioni molto piccole durante il shuffle, riducendo il numero di task effettivi. Pertanto l'effettivo numero di partizioni può cambiare in runtime per DF/SQL rimanendo invece rigido su RDD, portando a migliori

prestazioni del primo.

La Query 3 mostra risultati un po' diversi, le tre implementazioni DF_Approx, DF_Exact e SQL mostrano tempi simili tra loro, la leggera differenza tra DF_Approx (che usa strutture dati complesse come i t-digest) e DF_Exact (richiede ordinamento completo) è trascurabile, suggerendo che per questo dataset (viene applicato sui valori orari, avendo 2 nazioni solo 48 valori) le funzioni percentile approssimato e esatto hanno costi computazionali praticamente equivalenti. L'aspetto più significativo riguarda la versione RDD, che mantiene un vantaggio prestazionale rispetto alle versione DF/SQL per tutto l'intervallo di worker testato. Questa differenza può essere data dalla natura delle funzioni usate, infatti la funzione approx_percentile deve costruire una struttura di approssimazione che legge tutti i punti e li inserisce nel digest, per dataset molto piccoli questo lavoro può essere overkill, quindi la versione RDD dove i valori vengono raccolti in una lista, risulta in questo caso più performante. Ricordiamo che Spark è ottimizzato per l'utilizzo su grandi dataset, non è quindi ottimale per l'utilizzo su dataset più piccoli come il nostro dataset di studio.

VIII. Automatizzazione

L'intera pipeline può essere eseguita automaticamente tramite lo script Bash `start.sh`. Questo script verifica innanzitutto la presenza dei pacchetti necessari e, in caso contrario, provvede alla loro installazione. Successivamente avvia i container richiesti per l'esecuzione e lancia il Process Group di NiFi per il caricamento dei file su HDFS. Una volta completato il trasferimento, lo script arresta il Process Group e procede all'esecuzione delle query previste. Infine, richiama il Process Group incaricato di trasferire i risultati da HDFS a Redis.