Apache Spark per l'analisi dei dati energetici e ambientali di Electricity Maps

Massimo Buniy
Studente Ing. Inform., Tor Vergata
Matricola: 0350022
massimo.buniy@students.uniroma2.eu

***Sommario*—This document is a model and instructions for LaTeX. This and the IEEEtran.cls file define the components of your paper [title, text, heads, etc.]. *CRITICAL: Do Not Use Symbols, Special Characters, Footnotes, or Math in Paper Title or Abstract.**

***Index Terms*—component, formatting, style, styling, insert**

I. Introduzione

A. Contesto e motivazione

Negli ultimi anni, la crescente disponibilità di grandi volumi di dati ha reso essenziale la realizzazione di pipeline di elaborazione dati efficienti, scalabili e automatizzate. In ambito aziendale e scientifico, la capacità di acquisire, elaborare e visualizzare informazioni in tempo reale o quasi rappresenta un vantaggio competitivo. Strumenti come Apache NiFi, Hadoop, Apache Spark, InfluxDB e Grafana offrono soluzioni complementari per la gestione, analisi e rappresentazione di dati su larga scala.

Il progetto si inserisce in questo contesto, con l'obiettivo di realizzare una pipeline completa che consenta di gestire dati in ingresso, trasformarli, analizzarli e visualizzarli in modo integrato e automatizzato.

B. Obiettivi del progetto

Il progetto mira a sviluppare un sistema modulare e containerizzato in grado di:

- Acquisire dati da fonti esterne tramite Apache NiFi e salvarli su un sistema distribuito HDFS in formati CSV e Parquet;
- Eseguire analisi distribuite tramite Apache Spark richiamato direttamente da NiFi;
- Memorizzare i risultati delle analisi in InfluxDB, un database time series ottimizzato per dati temporali;
- Visualizzare i dati e i risultati attraverso dashboard interattive realizzate con Grafana.

Il sistema deve essere facilmente deployabile tramite Docker Compose, garantendo scalabilità, riusabilità e semplicità di gestione.

C. Struttura della relazione

La relazione è organizzata come segue: nella *sezione II* vengono descritti i requisiti funzionali e non funzionali, insieme ai vincoli del progetto. La *sezione III* illustra l'architettura del sistema, presentando i singoli componenti e il loro funzionamento integrato. La *sezione IV* tratta l'implementazione tecnica, la struttura del codice, la containerizzazione e l'automazione del flusso di lavoro. La *sezione V* serve a riportare i risultati ottenuti dall'esecuzione delle query. infine, la *sezione VI* valuta le prestazioni del sistema con riferimento a test effettuati.

II. Requisiti e Specifiche

A. Requisiti funzionali

Il sistema deve soddisfare i seguenti requisiti funzionali:

- Scaricare automaticamente il dataset dal sito di *Electricity Maps*.
- Effettuare l'ingestion dei dati tramite Apache NiFi, che deve salvare i dati grezzi in formato CSV all'interno di HDFS.
- I job di elaborazione devono essere eseguiti tramite Apache Spark, leggendo i dati direttamente da HDFS.
- Le elaborazioni devono rispondere alle query 1 e 2, come specificato nei requisiti, producendo i risultati attesi.
- I risultati dei job Spark devono essere scritti nuovamente su HDFS in formato CSV.
- I dati di output devono essere inoltre inseriti in InfluxDB per permettere la visualizzazione temporale tramite Grafana.
- Garantire l'orchestrazione e l'automazione del flusso dati, dall'ingestion fino alla visualizzazione.

B. Vincoli e assunzioni

Per la realizzazione del progetto sono stati considerati i seguenti vincoli e assunzioni:

- Il dataset fornito da *Electricity Maps* è disponibile in formato CSV e presenta una struttura dati coerente e stabile nel tempo. Inoltre, si assume che il link al download mantenga invariata la struttura e resti valido fino alla presentazione del progetto.
- Il sistema è progettato per essere eseguito in un ambiente di test basato su container Docker, con risorse hardware limitate rispetto a un cluster di produzione. In particolare, i test sono stati effettuati su una macchina con le seguenti caratteristiche:
  - Host: HP Pavilion Gaming Laptop 15-dk0xxx
  - Processore: Intel Core i7 9th Gen
  - Memoria fisica totale: 16 GB RAM
  - Sistema operativo: Microsoft Windows 11 Home (64-bit)

- Si assume che Apache NiFi possa accedere e scrivere correttamente i dati su HDFS senza problemi di permessi o connettività.
- Le elaborazioni Apache Spark sono eseguite con un numero limitato di nodi (uno master e due worker), sufficiente a garantire prestazioni accettabili per i dataset considerati.
- La sicurezza del sistema è garantita tramite credenziali statiche e non sono implementati meccanismi avanzati di autenticazione o autorizzazione.
- La frequenza di aggiornamento dei dati è limitata alla disponibilità del dataset sul sito esterno e non è gestita dinamicamente in tempo reale.
- La visualizzazione tramite Grafana si basa esclusivamente sui dati storicizzati in InfluxDB e non prevede analisi in tempo reale o interattive avanzate.

III. Architettura del Sistema

In questa sezione viene presentata l'architettura complessiva del sistema sviluppato, descrivendo i componenti principali, il loro ruolo e le modalità di interazione. L'architettura è progettata per supportare un flusso dati automatizzato che va dall'acquisizione dei dati grezzi fino alla visualizzazione dei risultati.

A. Panoramica generale

Il sistema è costituito da una pipeline di elaborazione dati basata su tecnologie open source che interagiscono tramite una rete containerizzata. I dati vengono acquisiti, trasformati, elaborati e infine visualizzati tramite una serie di servizi orchestrati tra loro. I principali componenti sono:

- **Apache NiFi**: responsabile dell'ingestion e della movimentazione dei dati, effettua il download del dataset da *Electricity Maps*, e li salva su HDFS sia in formato CSV che Parquet.
- **Hadoop (HDFS)**: sistema di storage distribuito utilizzato per conservare i dati grezzi e i risultati finali dei processi Spark.
- **Apache Spark**: piattaforma di calcolo distribuito utilizzata per eseguire le query di analisi sui dati presenti in HDFS.
- **InfluxDB**: database temporale dove vengono salvati i risultati aggregati delle elaborazioni per permettere la visualizzazione storica.
- **Grafana**: strumento di visualizzazione che si connette a InfluxDB per mostrare i dati in forma grafica e interattiva.

B. Descrizione dei componenti

1) Apache NiFi: Apache NiFi gestisce il flusso di dati in ingresso, automatizzando il processo di download, preelaborazione e salvataggio su HDFS. Utilizza processori dedicati per convertire i dati in formato Parquet, facilitando l'elaborazione successiva.

L'intero sistema è containerizzato utilizzando Docker e orchestrato tramite `docker-compose`.

Il servizio **Apache NiFi** è configurato tramite un Dockerfile dedicato, situato nella cartella `docker-NiFi`. La versione utilizzata è la 1.28.1, che integra nativamente i processori necessari per la comunicazione con Apache Hadoop. Nel Dockerfile, inoltre, viene installato il client Spark per consentire a NiFi di invocarlo direttamente.

Il container NiFi monta i volumi necessari, tra cui:

- La configurazione di Hadoop, montata da `./docker-NiFi/hadoop-conf`, che permette a NiFi di interagire correttamente con HDFS.
- Gli script Spark, montati dalla cartella `./scripts`, per automatizzare le elaborazioni.

Inoltre, il container espone la porta HTTPS 8443 per l'interfaccia web e utilizza variabili di ambiente per configurare la sicurezza in modalità single user.

Nella cartella `docker-NiFi` è inoltre presente un template NiFi da importare manualmente tramite l'interfaccia web di NiFi. Questo template contiene il flusso di dati preconfigurato che automatizza il processo di download, elaborazione e scrittura su HDFS ed avvio dei job Spark semplificando l'avvio del sistema.

2) Hadoop (HDFS): HDFS funge da sistema di storage distribuito, garantendo la persistenza dei dati sia grezzi sia elaborati. La scelta di HDFS consente di scalare facilmente il sistema in caso di aumento volumetrico dei dati.

Il sistema utilizza un cluster Hadoop configurato tramite `docker-compose`, composto da un nodo master e più nodi slave per garantire la distribuzione e la ridondanza dei dati. Nel file `docker-compose.yml` i container sono definiti come segue:

- `hadoop-master`: nodo master con NameNode e servizi principali, espone la porta web 9870 per la UI di gestione.
- `hadoop-slave1`, `hadoop-slave2`, `hadoop-slave3`: nodi DataNode che gestiscono la memorizzazione distribuita e replicata.

Tutti i container condividono la rete `hadoop_network` per permettere la comunicazione tra i nodi. Il Dockerfile personalizzato, basato su Ubuntu 18.04, installa Java 8, Hadoop 3.3.6 e configura SSH senza password per abilitare il funzionamento distribuito in modalità pseudo-distribuita. I file di configurazione Hadoop (ad esempio `core-site.xml`, `hdfs-site.xml`, `workers`, ecc.) sono aggiunti tramite la cartella `config` e caricati all'avvio tramite uno script di bootstrap (`bootstrap.sh`).

Questa architettura containerizzata consente di replicare facilmente l'ambiente di produzione in ambienti di test, garantendo scalabilità e persistenza dei dati tramite HDFS.

La struttura di Apache Hadoop all'interno di Docker è stata presa dal repository GitHub del dottor Nardelli. In particolare, si è scelto di adottare questa repository per avere tutti i Dockerfile all'interno di un unico progetto e per aggiornare la versione di Hadoop alla 3.3.6, poiché il download della versione 3.3.2 non era più disponibile.

3) Apache Spark: Apache Spark esegue i job di analisi dati parallelizzati, leggendo direttamente i dati da HDFS e scrivendo i risultati in formato CSV. Il sistema è configurato con un nodo master e due nodi worker, tutti containerizzati per simulare un cluster di elaborazione distribuita.

Nel file `docker-compose.yml` i container Spark sono definiti come segue:

- `spark-master`: nodo master, espone la porta 7077 per la comunicazione e 8080 per l'interfaccia web di monitoraggio.
- `spark-worker-1` e `spark-worker-2`: nodi worker che eseguono i task distribuiti, collegati al master tramite l'URL `spark://spark-master:7077`.

Tutti i container Spark condividono la rete `hadoop_network` per consentire la comunicazione con Hadoop e tra i nodi Spark. Viene inoltre montata la cartella `./scripts` all'interno dei container per rendere disponibili gli script Spark da eseguire.

Il Dockerfile personalizzato parte dall'immagine `bitnami/spark:3.5.6` e aggiunge l'installazione di `pip3` per Python 3, necessaria per gestire le dipendenze richieste dagli script di analisi. In particolare, vengono copiati e installati i pacchetti Python indicati nel file `requirements.txt`, situato nella cartella `scripts`.

Questa configurazione consente di eseguire in sicurezza gli script Spark con tutte le librerie necessarie, garantendo la possibilità di scrivere direttamente i risultati su database esterni come InfluxDB, utilizzato per la visualizzazione dei dati elaborati.

4) InfluxDB: InfluxDB è utilizzato per raccogliere e memorizzare i risultati aggregati delle query in forma di serie temporali. Questo consente di gestire efficientemente dati storici, permettendo analisi temporali avanzate e visualizzazioni in tempo reale tramite strumenti come Grafana.

Nel file `docker-compose.yml` il servizio InfluxDB è configurato come segue:

- L'immagine utilizzata è `influxdb:2.7`, una versione stabile e aggiornata.
- Il container espone la porta 8086, utilizzata per l'accesso HTTP all'API di InfluxDB.
- Viene montato un volume persistente `influxdb_data` per salvare i dati e mantenere la persistenza tra i riavvii.
- Il container è collegato sia alla rete `hadoop_network`, per consentire la comunicazione con i job Spark e NiFi, sia alla rete `grafana_network`, per consentire l'accesso a Grafana.
- Variabili d'ambiente inizializzano il database con:
  - modalità `setup` per la configurazione automatica all'avvio,
  - username e password amministrativi,
  - organizzazione (`ralisin`),
  - bucket di default (`init_bucket` ma non utilizzato effetticamente poiché la singola query si crea il proprio bucket),
  - token di amministrazione per l'autenticazione, necessario sia per la scrittura da Spark, ma anche per l'accesso in lettura su Grafana.

Questa configurazione consente un avvio rapido del servizio InfluxDB pronto all'uso nel sistema, integrandosi con il flusso di lavoro di elaborazione dati e visualizzazione.

5) Grafana: Grafana è utilizzato per la visualizzazione grafica dei dati salvati su InfluxDB. Fornisce dashboard interattive per monitorare le metriche e le tendenze derivanti dalle elaborazioni.

Nel file `docker-compose.yml` il servizio Grafana è configurato come segue:

- Utilizza l'immagine ufficiale `grafana/grafana`, garantendo aggiornamenti e supporto continuo.
- Il container espone la porta 3000, accessibile via browser per l'interfaccia web di Grafana.
- Viene montato un volume persistente `grafana_data` per conservare la configurazione delle dashboard e i dati di stato.
- Grafana dipende dal servizio InfluxDB, assicurando che il database sia attivo prima dell'avvio.
- È connesso alla rete `grafana_network` per comunicare con InfluxDB.

C. Schema architetturale

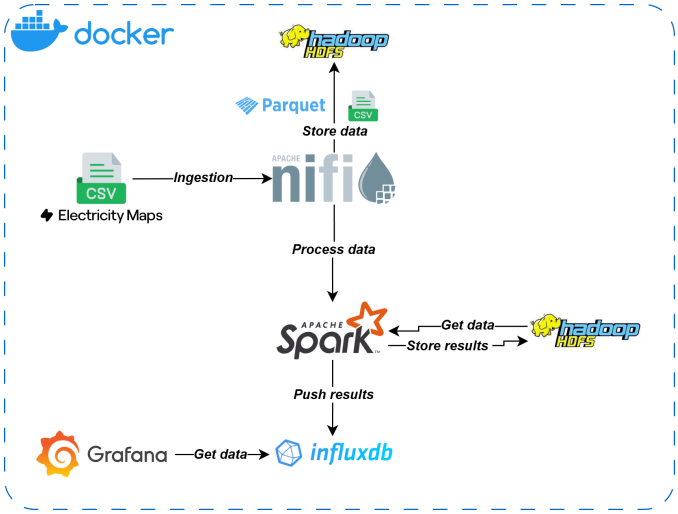

Figura 1. Schema architetturale

Lo schema in Figura 1 mostra la comunicazione tra i componenti del sistema, evidenziando il flusso dati dall'ingestion, all'analisi e visualizzazione.

D. Configurazione

Dopo l'avvio dei container tramite `docker-compose up`, sono necessari alcuni passaggi manuali per completare la configurazione del sistema.

a) Configurazione di Apache NiFi:

1) Importare il template presente nella cartella `./docker-NiFi` dalla schermata web di NiFi (porta 8443). Le credenziali per accedere a NiFi sono hardcoded nel docker-compose e sono:
   - user: admin
   - password: passwordpass
2) Inserire il template all'interno del canvas principale.
3) Avviare il controller service **CSVReader** (necessario per la conversione dei file CSV in Parquet).
4) Avviare i seguenti elementi del flusso:
   - il gruppo `Download & Rename CSV data hourly`
   - il processore `MergeContent`
   - il processore `Query 1 RDD CSV`
   - il processore `Query 2 RDD CSV`

Nota: sul template NiFi sono riportati anche altri sei processori. Questi possono essere utilizzati per eseguire in singolo le query e sono già impostati per lanciare le query con le varie configurazioni possibili (rdd o df, csv o parquet).

b) Configurazione di Grafana:

1) Collegare la sorgente dati **InfluxDB** da `http://influxdb:8086`, utilizzando le credenziali e il token configurati in `docker-compose.yml`.
   - INFLUXDB_URL = "http://influxdb:8086"
   - ORG = "ralisin"
   - TOKEN = "my-super-secret-token"
2) Importare la dashboard JSON disponibile nella cartella `./docker-grafana`.

IV. Implementazione

A. Organizzazione del codice e script

Il progetto è strutturato per essere facilmente manutenibile e modulare. Le directory principali sono organizzate come segue:

- `docker-NiFi`: contiene il Dockerfile e la configurazione per il container NiFi, oltre al template da importare manualmente nell'interfaccia di NiFi per avviare il flusso dati.
- `docker-hadoop`: include il Dockerfile per l'ambiente Hadoop, i file di configurazione (es. `core-site.xml`, `hdfs-site.xml`, `workers`) e lo script `bootstrap.sh` che inizializza i nodi NameNode e DataNode.
- `docker-spark`: contiene il Dockerfile per l'immagine Spark, basata su `bitnami/spark:3.5.6`. Installa Python e le dipendenze necessarie per eseguire i job Spark, inclusa la scrittura dei risultati su InfluxDB.
- `docker-grafana`: raccoglie eventuali file di provisioning o dashboard JSON da importare manualmente nell'interfaccia di Grafana.
- `scripts`: raccoglie gli script PySpark per l'elaborazione dei dati. Questi sono divisi per query, per cui all'interno della cartella saranno presenti i file: `query1.py` e `query2.py`. Inoltre vi è un file chiamato `Utils.py` che contiene una serie di funzioni comuni ad entrambe le query.

Il file `docker-compose.yml` definisce e orchestra i vari container, garantendo la corretta interconnessione tra i servizi tramite le reti Docker. Ogni componente (NiFi, Hadoop, Spark, InfluxDB, Grafana) è isolato in un container dedicato ma comunicante, con volumi condivisi dove necessario per scambiare file o mantenere dati persistenti.

1) Dettaglio degli scripts: Entrambe le query sono state implementate in linguaggio Python utilizzando PySpark. Gli script sono stati progettati in modo da essere configurabili tramite argomenti passati da linea di comando. In particolare, sono previsti tre argomenti principali:

- `--mode`: specifica la modalità di esecuzione della query. Può assumere i valori `rdd` oppure `df`, indicando se utilizzare l'API basata su RDD (Resilient Distributed Datasets) o quella basata su DataFrame.
- `--format`: definisce il formato dei dati da leggere da HDFS. È possibile scegliere tra `csv` e `parquet`. L'opzione `parquet` è supportata solamente in modalità `df`, poiché il formato Parquet è direttamente integrato nelle API DataFrame di Spark.
- `--save`: è un flag di tipo booleano. Se specificato, e se lo script viene eseguito in modalità `rdd`, i risultati ottenuti verranno salvati su InfluxDB, rendendoli disponibili per la visualizzazione tramite Grafana.

La scelta di realizzare script modulari e indipendenti nasce dalla necessità di confrontare le performance derivanti dalle varie combinazioni dei parametri `mode` e `format`. Durante lo sviluppo, è emerso che eseguire più combinazioni in sequenza all'interno dello stesso processo Python produceva risultati non affidabili, a causa del meccanismo di caching automatico di Spark. Per evitare che i dati venissero riutilizzati implicitamente dalle esecuzioni precedenti, si è deciso di lanciare ogni combinazione in un processo isolato, garantendo così misurazioni corrette e confrontabili.

2) Query 1:

a) Implementazione con RDD: La versione RDD della *Query 1* si basa sull'aggregazione per anno e zona geografica dei dati di intensità di carbonio e percentuale di energia carbon-free. Dopo aver unito in un singolo RDD tutti i file di input, si esegue un primo filtro per escludere eventuali record incompleti (controllo sovrabbondante, ma per scrupolo ho deciso di implementarlo). Ogni elemento dell'RDD viene poi trasformato in una coppia chiave-valore dove la chiave è la coppia `(anno, zona)` e il valore è una tupla contenente i due valori numerici di interesse: intensità di carbonio e percentuale CFE.

Successivamente, ogni valore viene mappato in una struttura che contiene: `(somma, minimo, massimo)` per ciascuna delle due metriche, più un contatore. Con un'operazione di `reduceByKey` viene calcolata l'aggregazione per ciascuna chiave, combinando le tuple intermedie. Il risultato viene infine

ordinato, trasformato in formato CSV e, opzionalmente, scritto su InfluxDB se lo script è stato lanciato con il flag `--save`.

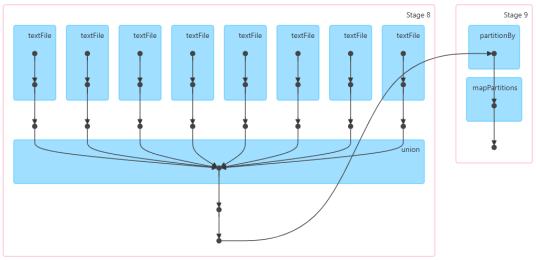

Figura 2. Q1 - DAG della soluzione con RDD

b) Implementazione con DataFrame: L'implementazione con DataFrame sfrutta le API SQL-like di Spark per ottenere un codice più conciso e performante, soprattutto su grandi volumi di dati. In base al parametro `--format`, i file vengono letti in formato CSV o Parquet. In quest'ultimo caso si applica anche una normalizzazione dei nomi delle colonne, necessaria per riuscire ad avere esattamente la stessa logica all'interno della funzione, e non dover scrivere una serie di controlli per cambiare il 'nome'.

Una volta caricato il DataFrame, si esegue un filtro per rimuovere i record con valori nulli nelle colonne necessarie (anche qui questo controllo da ipotesi è sovrabbondante, ma per scrupolo e per simmetria con l'implementazione rdd ho deciso di mantenerlo). Viene poi estratto l'anno dalla data e utilizzato per raggruppare i dati insieme al nome della zona. Si calcolano quindi, tramite le funzioni di aggregazione fornite da Spark, i valori medi, minimi e massimi per le due metriche considerate. Il risultato è ordinato per anno e zona e convertito infine in formato CSV per essere scritto su file o utilizzato in analisi successive.

Questa versione è più compatta, leggibile e generalmente più efficiente rispetto all'approccio RDD.

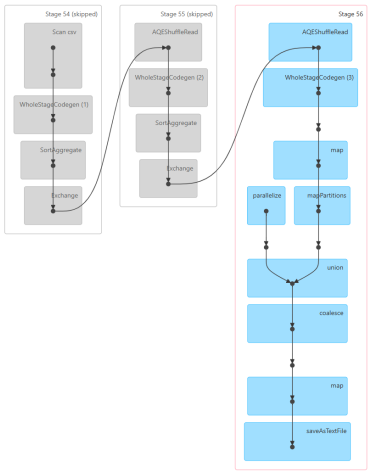

Figura 3. Q1 - DAG della soluzione con DF

3) Query 2:

a) Implementazione con RDD: Nella versione RDD della *Query 2*, l'obiettivo è calcolare, per ciascun mese, la media dell'intensità di carbonio e della percentuale di energia carbon-free, identificando inoltre i 5 mesi migliori e peggiori secondo ciascuna metrica. I dati vengono prima filtrati per rimuovere i record con valori nulli e successivamente trasformati, modificando la colonna data nel formato `yyyy_MM` (anno_mese).

Il mapping associa ogni mese ai valori numerici di interesse, che vengono poi aggregati tramite `reduceByKey` per ottenere somma e conteggio. Da qui si calcolano le medie mensili. Viene poi effettuato un ordinamento su questi valori medi per estrarre i 5 mesi con valori più alti e più bassi sia per intensità di carbonio che per percentuale CFE. Il risultato è restituito sia in forma completa (media mensile per ogni mese), sia in forma "ranked" per i soli top/bottom 5.

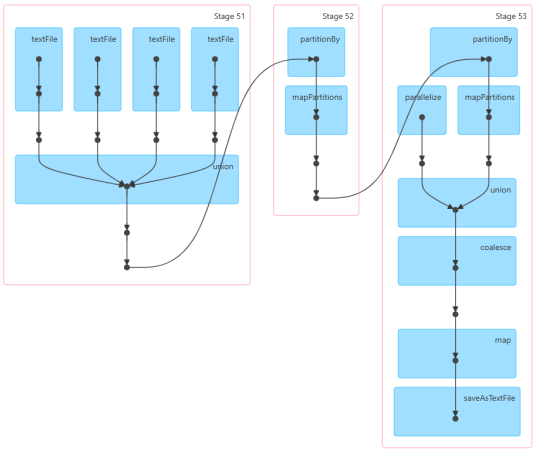

Figura 4. Q2 - DAG della soluzione con RDD

b) Implementazione con DataFrame: L'approccio basato su DataFrame utilizza funzioni Spark SQL per calcolare le stesse metriche in modo più conciso e ottimizzato. I file vengono letti in formato CSV o Parquet, con eventuale normalizzazione dei nomi delle colonne. Dopo un filtro iniziale, la colonna temporale viene convertita nel formato `yyyy_MM`, e si selezionano solo le colonne necessarie.

Il DataFrame risultante viene raggruppato per mese e aggregato per ottenere la media mensile di intensità di carbonio e percentuale CFE. Le operazioni di ordinamento e selezione permettono di ottenere facilmente i 5 mesi migliori e peggiori per ciascuna metrica. I risultati finali sono convertiti in RDD per il salvataggio su HDFS. Questo approccio risulta generalmente più performante rispetto alla versione RDD.

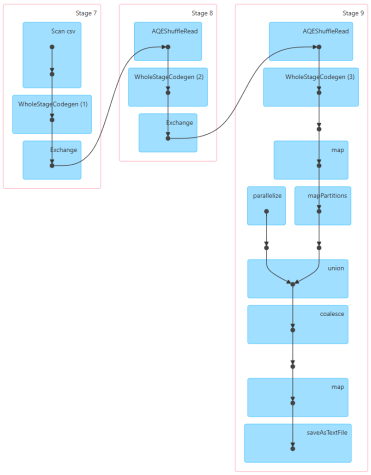

Figura 5. Q2 - DAG della soluzione con DF

V. Risultati ottenuti

A. Query 1

Il dataset analizzato riporta valori annuali, dal 2021 al 2024, relativi a due paesi europei, Italia e Svezia, confrontati con i valori medi, minimi e massimi a livello europeo per due indicatori chiave: l'intensità di carbonio (carbon intensity) e la percentuale di energia senza emissioni di carbonio.

1) Carbon intensity: Dai grafici emerge che la Svezia mantiene costantemente un livello molto basso di intensità di carbonio, significativamente inferiore alla media europea e decisamente sotto i valori massimi registrati. Questo riflette un sistema energetico più pulito e meno dipendente dai combustibili fossili.

L'Italia, al contrario, mostra valori di intensità di carbonio più elevati, pur con una tendenza generale alla diminuzione dal 2021 al 2024. Ciò suggerisce un percorso di decarbonizzazione in atto, anche se l'intensità rimane superiore e molto distante dai livelli della Svezia.

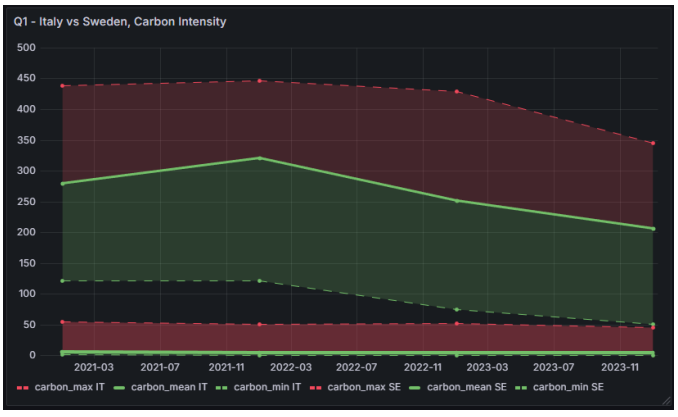

Figura 6. Carbon Intensity annuale, Italia vs Svezia

2) Carbon-free energy percentage: Nei grafici, la Svezia si distingue per valori di CFE% estremamente elevati, superiori al 90% in tutti gli anni considerati, evidenziando un'alta dipendenza da energia pulita.

L'Italia, pur mostrando un aumento costante del CFE% dal 2021 al 2024, parte da valori significativamente più bassi (intorno al 77% nel 2021) e si avvicina gradualmente al 90% nel 2024. Questo indica un miglioramento nella quota di energia carbon-free, ma con ancora margini di crescita rispetto alla Svezia.

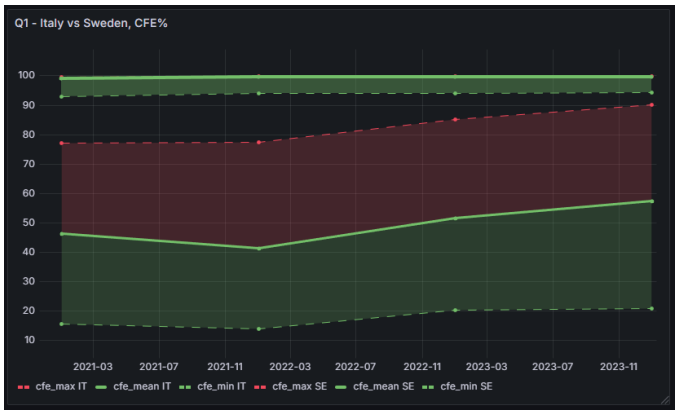

Figura 7. CFE% annuale, Italia vs Svezia

B. Query 2

Per l'Italia, oltre all'analisi annuale comparativa, è stata condotta un'analisi più approfondita su base mensile, relativa all'intensità di carbonio (carbon intensity) e alla percentuale di energia carbon-free (CFE%). Questi dati permettono di evidenziare variazioni stagionali, tendenze a breve termine e progressi continui nel mix energetico nazionale.

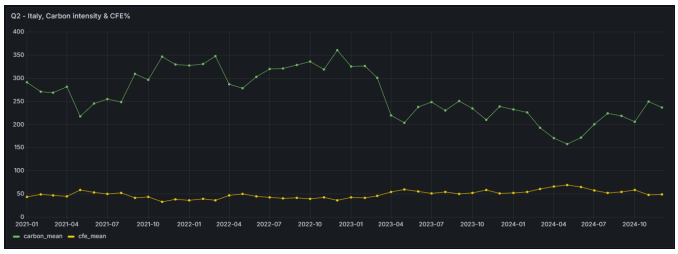

Figura 8. Carbon Intensity e CFE% mensile in Italia

1) Carbon intensity: Dal 2021 al 2024 si osserva una chiara tendenza alla diminuzione dell'intensità di carbonio su base mensile. Nel 2021 e nel 2022, i valori mensili oscillano tra circa 220 e 350 gCO_2/kWh, con picchi significativi nei mesi invernali e autunnali, probabilmente legati a una maggiore domanda energetica e a un maggiore ricorso a fonti fossili. Dal 2023 in poi si evidenzia un miglioramento: i valori si stabilizzano su livelli inferiori rispetto agli anni precedenti, con minimi intorno ai 160–170 gCO_2/kWh nei mesi primaverili del 2024, segno di un'evoluzione del mix energetico verso fonti più sostenibili.

Figura 9. Carbon Intensity mensile in Italia

C. Carbon-free energy percentage

Parallelamente, la quota di energia carbon-free mostra un miglioramento continuo:

- Nel 2021, i valori si attestano tra 33% e 58%, con un andamento piuttosto irregolare.
- Nel 2022, la CFE% rimane sotto il 50% per la maggior parte dell'anno.
- A partire dal 2023 si registra un netto incremento, culminando nel 2024 con mesi che superano il 65%, in particolare in primavera, quando la produzione da rinnovabili è probabilmente più elevata.

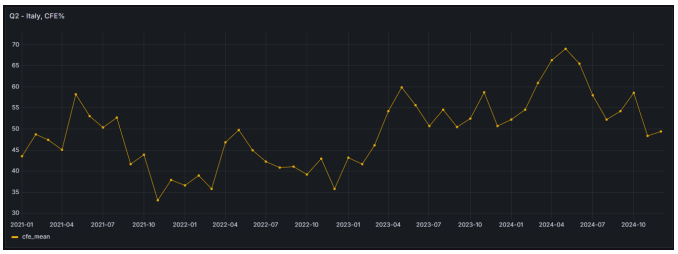

Figura 10. CFE% mensile in Italia

D. Analisi conclusiva

L'analisi mensile evidenzia una stagionalità ben marcata:

- Nei mesi primaverili (aprile-maggio), l'intensità di carbonio tende a essere più bassa e la percentuale di energia carbon-free più alta.
- Al contrario, nei mesi invernali (novembre-gennaio) l'intensità di carbonio aumenta, mentre la CFE% cala, suggerendo una maggiore dipendenza da fonti fossili durante i periodi di picco di consumo.

L'Italia mostra, nel quadriennio analizzato, un miglioramento costante e significativo nella sostenibilità del proprio sistema energetico. La riduzione dell'intensità di carbonio e l'aumento della quota di energia carbon-free, soprattutto nel 2023 e nel 2024, indicano progressi concreti nella transizione energetica, pur restando evidenti le sfide legate alla stagionalità e all'uso dei combustibili fossili nei mesi più critici.

VI. Valutazione delle Prestazioni

A. Metodologia

Per valutare le prestazioni delle varie implementazioni della query, sono stati misurati i tempi di esecuzione in secondi utilizzando Python. Ogni variante della query è stata eseguita 10 volte, escludendo la prima esecuzione (più influenzata da overhead di inizializzazione) e calcolando la media sui restanti nove tempi. Le implementazioni testate sono:

- `query1_rdd_csv`: utilizzo di `RDD` su file CSV
- `query1_df_csv`: utilizzo di `DataFrame` su file CSV
- `query1_df_parquet`: utilizzo di `DataFrame` su file Parquet
- `query2_rdd_csv`: utilizzo di `RDD` su file CSV
- `query2_df_csv`: utilizzo di `DataFrame` su file CSV
- `query2_df_parquet`: utilizzo di `DataFrame` su file Parquet

Per ogni variante, l'esperimento è stato condotto sia per l'Italia (IT) che per la Svezia (SE) per la query 1, e per solo l'Italia (IT) per la query 2.

B. Risultati

I risultati ottenuti sono riportati nella Tabella I per quel che riguarda la query 1 e Tabella II per quel che riguarda la query 2. Si evidenzia che:

- L'uso del formato `Parquet` è sistematicamente più veloce rispetto al formato `CSV`, grazie alla maggiore efficienza nell'accesso ai dati e alla compressione.
- Le implementazioni basate su `DataFrame` offrono prestazioni migliori rispetto a quelle basate su `RDD`, in entrambi i formati.

Implementazione	**IT (s)**	**SE (s)**
RDD + CSV	3.344	3.182
DataFrame + CSV	1.317	0.605
DataFrame + Parquet	1.091	0.498

Tabella I
TEMPI MEDI DI ESECUZIONE DELLA QUERY 1

Implementazione	**IT (s)**
RDD + CSV	13.800
DataFrame + CSV	1.487
DataFrame + Parquet	1.583

Tabella II
TEMPI MEDI DI ESECUZIONE DELLA QUERY 2

Riferimenti Bibliografici

[1] "matnar/docker-hadoop," GitHub. Accessed: May. 14, 2025. [Online]. Available: https://github.com/matnar/docker-hadoop