SABD

Christia Santapaola
Università di Roma Tor Vergata

Roma, Italia
9669chris@gmail.com

***Abstract*—**

I. Introduzione

Il presente progetto si propone di effettuare una serie di analisi batch su un insieme di dati finanziari. Tale compito viene realizzato mediante l'utilizzo dello stack tecnologico costituito da Apache Spark nella versione 3.4.0, il quale fa affidamento su un cluster Hadoop Distributed File System (HDFS) per l'esecuzione delle analisi. Al fine di effettuare l'ingestione dei dati su HDFS, si è fatto ricorso ad Apache Flume, un framework appartenente alla famiglia Apache dedicato alla gestione di flussi di dati. Infine, i risultati delle analisi vengono archiviati su HDFS nel formato CSV e contemporaneamente su un server Redis tramite l'ausilio di Spark.

II. Infrastruttura

Il software è stato implementato su un'architettura distribuita emulata mediante l'utilizzo di Docker. L'infrastruttura comprende diversi componenti, tra cui:

- Un cluster Hadoop Distributed File System (HDFS), composto da un namenode e una serie di datanode.
- Un cluster Spark, che include un master Spark e diversi worker Spark.
- Un server Redis.
- Un agente Apache Flume.

Tutti questi container sono collegati alla stessa rete, consentendo una comunicazione diretta tra di loro.

III. Data Ingestion

il termine "data ingestion" si riferisce al processo di acquisizione e importazione dei dati all'interno del sistema distribuito da diverse fonti esterne. Queste fonti potrebbero essere database, file di log, dispositivi di sensori, servizi web o altri sistemi di archiviazione dati. L'obiettivo della data ingestion è quello di raccogliere, organizzare e rendere disponibili i dati per l'elaborazione all'interno del file system distribuito. Durante la fase di data ingestion, i dati vengono letti dalle loro origini, trasformati, se necessario, e caricati nel file system distribuito in modo che possano essere accessibili e utilizzabili da altre applicazioni o processi all'interno del sistema. La data ingestion può comportare la trasformazione dei dati in un formato compatibile con il file system distribuito, la pulizia e l'arricchimento dei dati per migliorarne la qualità o la struttura, nonché la gestione delle operazioni di caricamento parallelo per consentire una rapida importazione dei dati. Una volta completata la fase di data ingestion, i dati saranno pronti per essere elaborati e/o analizzati. Nel progetto sono state usate due metodologie: La prima è stata la copia del file csv tramite comando hdfs; questo metodo è fragile e manuale, nel caso del progetto un solo file CSV è stato analizzato, quindi questo metodo è stato considerato sufficiente per lo scope, tuttavia in un progetto più esteso non è il metodo consigliato. Il secondo è stato tramite Apache Flume: Quest ultimo è un framework di data ingestion il cui obiettivo principale è semplificare la pipeline di dati, consentendo agli utenti di raccogliere e spostare i dati da sorgenti eterogenee verso destinazioni specifiche. Nel progetto è stato usato una container apposito per apache Flume in cui è stata utilizzata una sorgente di tipo 'spooldir'; definita una cartella sulla macchina locale, Apache Flume si occupa di scansionare la cartella per ogni file presente, spacchettarlo in un pacchetti di dimensioni configurabile è di inviarlo a tutte le destinazioni configurate. Come destinazione (sink) è stato usato HDFS stesso, con una dimensione di pacchetto di 25MB, allo scopo di migliorare il throughput del trasferimento. Apache Flume si assicura che il trasferimento vada a buon fine, di segnalare errori ritrovati, e garantisce automatismo e robustezza all'intero processo. Le sue principali debolezze sono:

- Il trasferimento non è real-time.
- Non supporta molto bene topologie dinamiche.

Tuttavia per il progetto realizzato, queste debolezze non sono state rilevanti.

IV. Struttura del dataset

Il nostro dataset è un file CSV contenente molti campi, di cui tuttavia a noi interessano solo 4:

- Il campo ID della forma NAME.NATION, il quale indica il nome dell'indice finanziario e la sua nazionalità.
- IL campo SecType, un codice di sicurezza.
- Il campo Data, che indica la data dell'aggiornamento del dato di Prezzo, diviso in giorno e ora.
- Il campo Last, che indica l'ultimo prezzo registrato dell'indice alla data specificata.

V. Query 1

La prima interrogazione ha richiesto l'impiego del framework Apache Spark per effettuare una procedura di aggregazione dei valori medi, minimi e massimi di un campo prezzo relativo a un subset specifico di indici finanziari. Nello

specifico, si è trattato degli indici di nazionalità francese che presentavano il codice di sicurezza "E".

La struttura della query è stata organizzata secondo i seguenti passaggi sequenziali:

Fase di caricamento e pulizia: I dati, originariamente immagazzinati in un file CSV, sono stati acquisiti da Spark e sottoposti a una fase preliminare di pulizia. Durante tale operazione, si è provveduto all'eliminazione delle righe contenenti commenti o caratterizzate da una mancanza di informazioni significative. Inoltre, le stringhe presenti nel dataset sono state sottoposte a un processo di parsing per consentire una più agevole manipolazione dei dati.

Fase di filtraggio: Il primo passaggio operativo ha riguardato l'applicazione di un filtro finalizzato all'isolamento del sottogruppo di indici finanziari di interesse. Sono stati mantenuti soltanto gli indici aventi un ID riconducibile alla nazionalità francese, ossia quelli caratterizzati dalla terminazione "Nome.FR", e associati al codice di sicurezza "E". Tale operazione ha permesso di ottenere un RDD contenente esclusivamente i dati rilevanti per l'analisi.

Fase di raggruppamento: Successivamente, è stata eseguita un'operazione di raggruppamento (GroupBy) al fine di organizzare i dati in base a specifiche chiavi. Inizialmente, il dataset è stato trasformato in una struttura di tipo chiave-valore, in cui ciascun record è stato associato a una chiave rappresentata dall'ID concatenato alla data nel formato giorno+ora. Successivamente, è stata applicata l'operazione di raggruppamento effettiva, che ha consentito di aggregare tutti i valori del campo Last relativi ad ogni ID e ad ogni ora specifica.

Fase di calcolo dei valori statistici: Dopo aver completato la fase di raggruppamento, sono stati calcolati i valori statistici richiesti, ovvero il valore minimo, massimo e medio per ciascun raggruppamento. Inoltre, è stato tracciato il numero di eventi utilizzati per ogni calcolo effettuato, al fine di fornire un'indicazione sulla robustezza dei risultati ottenuti.

In tal modo, l'interrogazione è stata strutturata con meticolosità al fine di ottenere i risultati desiderati in modo accurato e significativo.

VI. Query 2

La seconda query richiedeva il calcolo della differenza oraria dei prezzi e la successiva determinazione della media e della deviazione standard di tali differenze su base giornaliera. Inoltre, era necessario individuare le cinque azioni con la peggiore diminuzione oraria e le cinque azioni con la migliore crescita oraria.

La query è stata progettata seguendo il seguente approccio metodologico:

Fase di Caricamento e Pulizia: I dati sono stati caricati su Spark e inizialmente è stata eseguita una fase di pulizia preliminare. Durante questa fase, sono state eliminate le righe commentate e le righe vuote presenti nel file CSV. Successivamente, le stringhe sono state parsate e trasformate in una struttura dati più comoda per facilitare l'accesso ai valori.

Prima Trasformazione: I dati sono stati ulteriormente elaborati come segue: innanzitutto, sono stati rimossi i dati duplicati in modo che ogni ID e ogni data avessero un solo record associato. Successivamente, i record sono stati trasformati in un formato di tipo chiave-valore, in cui la chiave rappresentava l'ID e il valore consisteva in una coppia (Data, Last).

Trasformazione dei dati in formato orario: I dati sono stati trasformati in un formato in cui a ciascun ID e a ciascuna data giornaliera veniva associato il valore dell'azione per ogni ora. Questo valore è stato calcolato prendendo l'ultimo valore noto dell'ora precedente. Nel caso in cui non ci fossero valori per un'ora specifica, ciò indicava che non vi erano state variazioni durante tale intervallo di tempo, pertanto si è ripetuto l'ultimo valore noto.

Calcolo delle differenze orarie: Successivamente, sono state calcolate le differenze orarie tra i valori delle azioni.

GroupBy e calcolo delle statistiche: Le differenze orarie sono state sottoposte a un'operazione di raggruppamento (GroupBy) utilizzando come chiave l'ID e la data giornaliera. Successivamente, sono state calcolate le diverse statistiche richieste, quali la media e la deviazione standard delle differenze orarie.

Infine, utilizzando l'RDD contenente le differenze orarie, sono state individuate le cinque azioni con la migliore performance e le cinque azioni con la peggiore performance. Il procedimento per entrambi i casi è analogo, differenziandosi solo per la logica di ordinamento. Prima di tutto, le differenze orarie sono state filtrate, prendendo per ogni coppia (ID, Data Giornaliera) Il valore migliore di Differenza, questo per evitare di avere ID duplicati nella classifica. Infine sono state sottoposte a un'operazione di raggruppamento (GroupBy) utilizzando come chiave la data giornaliera e una Tupla (ID, Differenza), sono state ordinate per Il valore della differenza e sono stati presi i primi 5 valori dell'ordinamento.

VII. Salvataggio CSV su HDFS

Per salvare i risultati delle query precedenti in formato CSV tramite Spark su HDFS, è stato adottato un processo diretto che ha seguito i seguenti passaggi:

1) La struttura RDD contenente i risultati è stata presa in considerazione.
2) È stata eseguita una funzione map() per trasformare i dati in formato CSV.
3) Successivamente, è stata eseguita un'operazione di coalesce(1) per raggruppare tutti i risultati in una singola unità parallela, al fine di semplificare il processo di salvataggio, infatti senza, ogni unita parallela di Spark scrive la sua parte in un file separato, questo comando permette di avere un singolo file.
4) Infine, i dati sono stati salvati su HDFS utilizzando il metodo appropriato fornito da Spark, 'saveAsTextFile()'

Attraverso questo approccio diretto e sequenziale, i risultati delle query sono stati adeguatamente formattati in formato CSV e salvati su HDFS, garantendo l'accessibilità e la gestione efficiente delle informazioni ottenute.

VIII. Salvataggio su Redis

Per il salvataggio su Redis, è stato utilizzato Redis for Spark, una libreria di integrazione specifica per il collegamento tra Spark e Redis. Questa libreria ha consentito di salvare i dati su Redis utilizzando un RDD di tipo ¡String, String¿ come input.

Durante il processo di salvataggio su Redis, è stata prestata particolare attenzione alla gestione delle chiavi duplicate al fine di evitare sovrascritture indesiderate. Pertanto, è stato necessario assegnare a ciascun record una chiave univoca per garantire l'integrità dei dati salvati.

La strategia adottata per il salvataggio su Redis è stata quella di raggruppare i valori in base all'ID dell'azione. In questo modo, tutti i risultati associati a un determinato ID sono stati inseriti in una stringa JSON. Questa semplice strategia ha permesso di evitare collisioni di chiavi, consentendo una corretta memorizzazione e recupero dei dati su Redis.

Grazie all'utilizzo di Redis for Spark e all'approccio strategico adottato per la gestione delle chiavi, il salvataggio dei risultati delle query su Redis è stato eseguito in modo affidabile e efficiente, garantendo l'integrità dei dati e la loro disponibilità per ulteriori elaborazioni o accessi successivi.

IX. Metriche di prestazioni

Per valutare le prestazioni delle query, sono state utilizzate le funzionalità di monitoraggio fornite dal servizio Spark spark-history. Durante l'esecuzione dell'applicazione tramite Spark-submit, sono stati specificati i parametri necessari per salvare i log dell'applicazione su HDFS. Successivamente, è stato configurato spark-history per acquisire i log dalla cartella di destinazione su HDFS, consentendo l'accesso tramite un'interfaccia web dedicata.

Questo approccio ha consentito di registrare e analizzare in modo dettagliato le metriche di esecuzione delle query, inclusi i tempi di elaborazione, l'utilizzo delle risorse e altre informazioni pertinenti. L'interfaccia web fornita da spark-history ha facilitato l'accesso a tali informazioni, consentendo un'analisi approfondita delle prestazioni dell'applicazione.

L'utilizzo di spark-history e la possibilità di esaminare i log tramite l'interfaccia web hanno fornito i risultati presenti nelle immagini 1 e 2

È fondamentale sottolineare che i risultati ottenuti dalle query eseguite nell'ambiente simulato su Docker non devono essere considerati come rappresentativi di un ambiente di esecuzione parallela reale. Nella realtà, la presenza di tempi di comunicazione non trascurabili e una rete effettiva potrebbero portare a risultati di prestazioni significativamente diversi da quelli attualmente registrati.

È emerso, tuttavia, che la fase di scrittura su HDFS rappresenta il principale ostacolo in termini di prestazioni. Tale operazione si è dimostrata notevolmente più onerosa rispetto alle altre attività svolte nell'applicazione.

In conclusione, mentre i risultati attuali derivanti dall'ambiente simulato forniscono una base di riferimento, è necessario considerare attentamente le implicazioni di un ambiente di esecuzione parallela reale e adottare strategie di

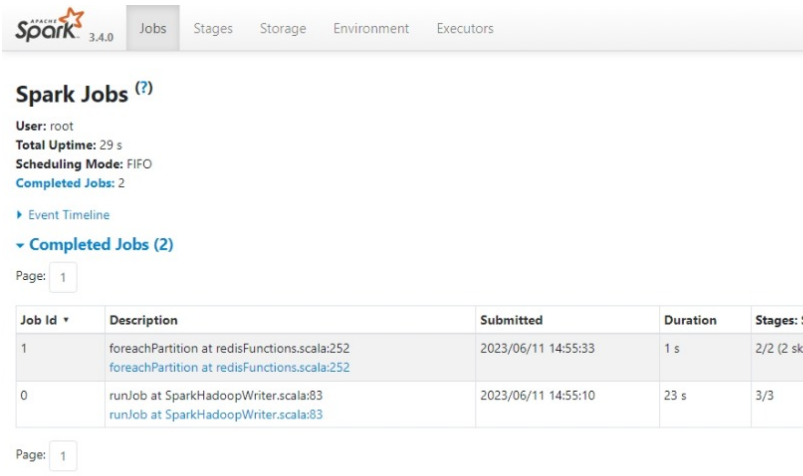

Fig. 1. Metriche per query 1

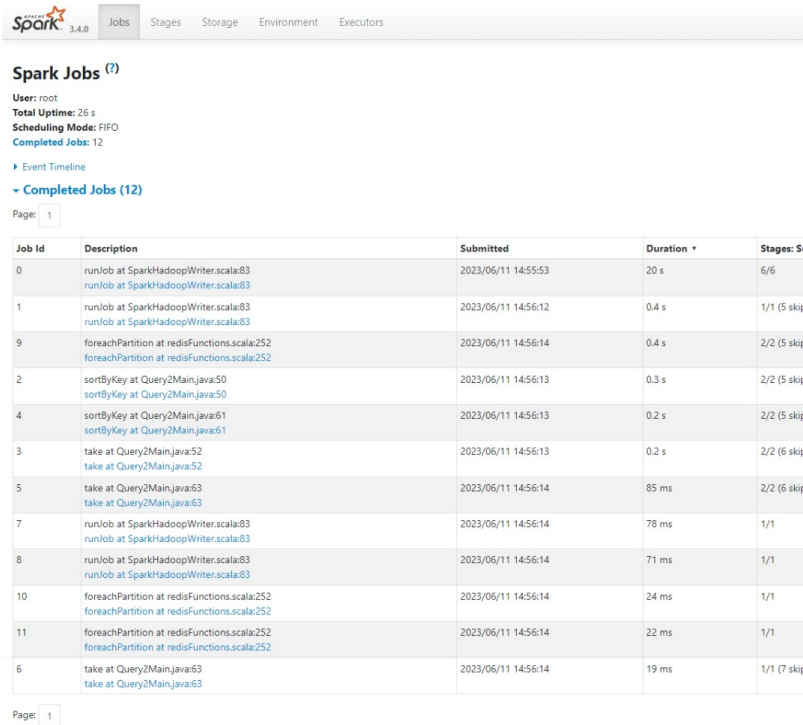

Fig. 2. Metriche per query 2

ottimizzazione appropriate per affrontare le sfide specifiche della scrittura su HDFS e migliorare le prestazioni complessive del sistema.

X. Conclusioni