Batch Processing con Spark

Primo progetto di Sistemi e Architetture per Big Data anno 2023/2024

Dissan Uddin Ahmed
0334869
Università degli studi di Roma
Tor Vergata
Roma, Italia
DissanAhmed@gmail.com

***Abstract*— Questo documento si pone lo scopo di spiegare il metodo adoperato e lo stack architetturale utilizzato per rispondere a 3 query assegnate nel primo progetto del corso di" Sistemi e Architetture per Big Data" della facoltà di Ingegneria Informatica magistrale dell'università di Roma Tor Vergata**

I. INTRODUZIONE

Lo scopo del progetto `e usare il framework di data processing Apache Spark per rispondere ad alcune query su dati di telemetria di circa 200k hard disk nei data center gestiti da Backblaze.

Il lavoro è suddiviso nelle seguenti fasi principali::

- Progettazione dell'Architettura
- Preprocessamento dei dati
- Analisi e svolgimento delle query
- Analisi delle prestazioni

Nel progetto si utilizzano i seguenti framework:

- Apache Nifi per la fase di ingestion e preprocessing
- Apache Hadoop il modulo HDFS come file system distribuito
- Apache Spark per il processamento dei dati
- Apache Cassandra per salvare i risultati su un database noSql

Il sistema è sviluppato in maniera distribuita mediante la tecnologia dei container usando l'ambiente di Docker, Docker Compose è l'orchestratore dei nodi.

II. DATASET

Per gli scopi di questo progetto, viene fornita una versione ridotta del dataset indicato nel Grand Challenge della conferenza ACM DEBS2024. Il dataset riporta i dati di monitoraggio S.M.A.R.T.2, esteso con alcuni attributi catturati da Backblaze. Il dataset contiene eventi riguardanti circa 200k hard disk, dove ogni evento riporta lo stato S.M.A.R.T. di un particolare hard disk in uno specifico giorno. Il dataset ridotto contiene circa 3 milioni di eventi (a fronte dei 5 milioni del dataset originario). In questo progetto sono stati considerati i seguenti campi:

Date: campo data nel seguente formato, yyyy-mm-ddThh:MM:ss.SSSSSS, riguarda la data della misurazione.

- Serial_number: stringa che rapresenta il singolo disco.
- Model: stringa che individua la casa produttrice.
- Failure: booleano indica se un disco ha subito un fallimento in un determinato giorno.
- Vault_id: intero rappresenta un gruppo di dischi.
- S9_power_on_hours: rappresenta il tempo per cui un disco e stato in funzione senza fallimenti.

III. ARCHITETTURA

L'architettura del sistema è realizzata utilizzando Docker Compose fig.1, che permette di orchestrare vari servizi all'interno di container, connessi tra loro attraverso una rete interna. Questa configurazione garantisce isolamento, scalabilità e facilità di gestione. Ogni servizio espone specifiche porte all'esterno per consentire l'accesso alle interfacce dei vari componenti. Di seguito viene descritto in dettaglio ogni componente dell'architettura fig1.

Apache NiFi: è utilizzato per la fase di ingestion e preprocessing dei dati. In questo progetto, per semplicità, NiFi è composto da un singolo nodo. Questo nodo è responsabile di raccogliere i dati di telemetria dagli hard disk, eseguire le prime trasformazioni necessarie e trasferire i dati preprocessati nel sistema HDFS. La scelta di un singolo nodo è giustificata dalla sufficiente capacità di NiFi di gestire il carico di lavoro previsto.

Apache Hadoop HDFS: HDFS (Hadoop Distributed File System) è utilizzato come file system distribuito per la memorizzazione dei dati. La configurazione di HDFS include:

- Namenode (Master): Gestisce il namespace del file system e regola l'accesso ai file da parte dei client.
- Datanode (Worker): Tre nodi worker sono impiegati per memorizzare effettivamente i dati. È stato scelto un grado di replicazione di 3 per garantire la ridondanza e la tolleranza ai guasti.
- ResourceManager: Coordina l'allocazione delle risorse in tutto il cluster.
- NodeManager: Gestisce le risorse e le applicazioni su ogni singolo nodo worker.

HDFS funge da intermediario tra il layer di ingestion (NiFi) e il layer di processamento (Spark), memorizzando i dati preprocessati pronti per l'analisi successiva.

Apache Spark: è utilizzato per il processamento dei dati. La configurazione di Spark include tre componenti chiave:

- Master Spark: Coordina l'esecuzione dei job di Spark.
- Worker Spark: Esegue le operazioni di processamento sui dati distribuiti.
- History Server: Fornisce un resoconto dettagliato delle prestazioni delle applicazioni Spark eseguite, permettendo di analizzare e ottimizzare i job di processamento.

Apache Cassandra: è utilizzato come datastore NoSQL per memorizzare i risultati delle query. La configurazione di Cassandra comprende tre nodi, che offrono scalabilità lineare e alta disponibilità. La distribuzione dei dati su più nodi garantisce prestazioni elevate e tolleranza ai guasti.

L'intero sistema è containerizzato utilizzando Docker, che assicura portabilità e isolamento tra i vari componenti. Docker Compose è impiegato per orchestrare i container, facilitando la gestione delle dipendenze e la configurazione dei servizi. Ogni componente del sistema è definito come un servizio all'interno del file docker-compose.yml, con le seguenti configurazioni di rete:

Rete Interna: Tutti i container sono connessi attraverso una rete interna, che permette una comunicazione efficiente e sicura tra i servizi.

Porte Esposte: Ogni servizio espone specifiche porte all'esterno, consentendo l'accesso alle interfacce di gestione e monitoraggio.

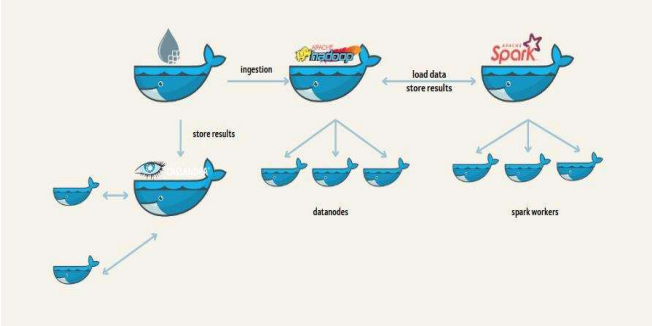

Fig. 1. Docker compose

IV. PREPROCESSAMENTO - NIFI

Il preprocessamento dei dati è realizzato utilizzando il framework Apache NiFi. Questo componente mette a disposizione un ambiente GUI accessibile dal browser, che permette di impostare e gestire i flussi di dati utilizzando i componenti denominati Processor. Di seguito viene descritta la pipeline di preprocessamento implementata in NiFi fig 2, fig3.

Descrizione del Flusso di Lavoro in NiFi

Scaricamento del Dataset: Il dataset di telemetria degli hard disk viene scaricato utilizzando il Processor InvokeHTTP, che consente di effettuare richieste HTTP per recuperare i file dai link forniti. Questa operazione automatizza il download dei dati necessari per l'analisi.

Gestione dei File Compressi: Il file scaricato è in formato tar, è stato necessario estrarne il contenuto per accedere ai dati grezzi. Questo viene realizzato utilizzando due Processors:

- CompressContent: Utilizzato per gestire i file compressi, eseguendo operazioni come la compressione e la decompressione dei file.
- UnpackContent: Utilizzato per estrarre il contenuto del file tar, trasformando il file compresso in un formato accessibile per ulteriori elaborazioni.

Conversione del Formato dei Dati: Il FlowFile generato dalla decompressione viene passato al gruppo di processamento. Per migliorare le prestazioni, il FlowFile viene convertito dal formato CSV a Parquet, un formato colonnare ottimizzato per le prestazioni di lettura e scrittura in ambiente Big Data. Questa conversione è effettuata per sfruttare i vantaggi di Parquet in termini di compressione e velocità di accesso ai dati.

Filtraggio delle Colonne: Le colonne del dataset vengono filtrate utilizzando il Processor QueryRecord, che permette di eseguire query SQL-like sui dati per selezionare solo le colonne necessarie per l'analisi. Questa operazione riduce la dimensione dei dati e facilita le operazioni successive di processamento.

Salvataggio dei Dati:I dati preprocessati, sia in formato CSV che Parquet, vengono salvati in HDFS per l'uso successivo da parte di Apache Spark. I dati sono memorizzati in due formati per garantire la compatibilità e ottimizzare le prestazioni delle query di analisi.

I dati preprocessati sono disponibili nel percorso HDFS /user/spark/pipeline/preprocessed. I template per creare il flusso di preprocessamento sono disponibili nella cartella locale nifi/flow-files. Questi template possono essere importati nell'ambiente NiFi per replicare o modificare il flusso di lavoro.

Diagrammi di Flusso:

Fig 2: Diagramma del flusso di lavoro di NiFi che mostra i Processors utilizzati per il download, decompressione e conversione dei dati.

Fig 3: Diagramma del flusso di lavoro di NiFi che illustra il filtraggio delle colonne e il salvataggio dei dati in HDFS.

Questa sezione approfondita descrive come NiFi viene utilizzato per automatizzare e ottimizzare il preprocessamento dei dati, assicurando che i dati grezzi siano trasformati e pronti per l'analisi successiva in Apache Spark.

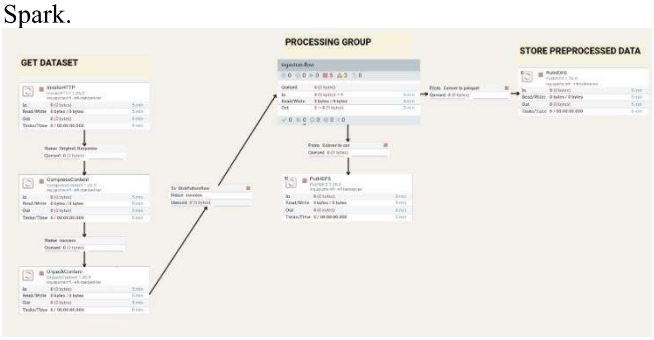

Fig. 2. Input-out nifi flow

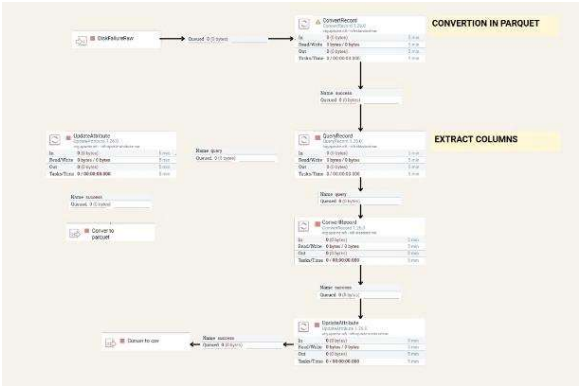

Fig. 3. Nifi-flow preprocessing.

V. PROCESSAMENTO - SPARK

Il processamento dei dati è stato eseguito utilizzando Apache Spark, sfruttando il linguaggio di programmazione Java per scrivere le query necessarie. In particolare, sono state utilizzate le API della libreria spark per i DataFrame per eseguire le operazioni di analisi sui dati di telemetria. Questa sezione fornisce una descrizione dettagliata del metodo impiegato per il processamento dei dati e l'osservazione delle prestazioni.

Linguaggio e API Utilizzati: Per scrivere le query, è stato utilizzato il linguaggio di programmazione Java, scelto per la sua robustezza e per le ampie librerie di supporto disponibili. Le API di spark.sql DataFrame sono state impiegate per le seguenti ragioni:

- Facilità d'uso: Le API di DataFrame offrono un'interfaccia di alto livello per lavorare con dati strutturati e semi-strutturati, facilitando la scrittura di query complesse.
- Prestazioni: DataFrame sfrutta l'ottimizzazione del Catalyst optimizer di Spark, migliorando l'efficienza delle operazioni di query.
- Compatibilità: Le API di spark.sql sono compatibili con diverse sorgenti di dati, inclusi formati CSV e Parquet, permettendo un facile accesso e processamento dei dati.

Dataset e Formati di Input: Ogni query è stata eseguita utilizzando il dataset in due formati distinti:

- Formato CSV: Un formato di testo semplice che memorizza i dati in righe, con ciascuna colonna separata da una virgola. Questo formato è ampiamente utilizzato e supportato, ma può essere meno efficiente in termini di prestazioni rispetto ad altri formati ottimizzati.
- Formato Parquet: Un formato colonnare che offre significativi vantaggi in termini di compressione e velocità di accesso ai dati. Parquet è ottimizzato per carichi di lavoro di analisi e può migliorare significativamente le prestazioni delle query.

L'esecuzione delle query su entrambi i formati ha permesso di osservare e confrontare le differenze di prestazioni tra i due formati, fornendo informazioni sull'efficienza di processamento.

Salvataggio dei Risultati: I risultati delle query sono stati salvati sia in HDFS che in una cartella locale per garantire l'accessibilità e la persistenza dei dati, i risultati sono stati salvati nella cartella di HDFS al percorso hdfs:namenode:54310/user/spark/pipeline/results/. Questo garantisce che i dati siano distribuiti e replicati, fornendo tolleranza ai guasti e alta disponibilità.

A. Query 1

L'obiettivo della prima query è determinare, per ogni giorno e per ogni vault, la lista dei vault che hanno subito esattamente 4, 3 o 2 fallimenti. Questo tipo di analisi è cruciale per identificare i vault con problemi ricorrenti e valutare la loro affidabilità nel tempo fig4. Di seguito Procedura di Implementazione:

- Selezione delle Colonne Rilevanti: Sono state selezionate solo le colonne necessarie per l'analisi: data, vault_id e failure. Questo passaggio riduce la quantità di dati da elaborare, migliorando l'efficienza della query.
- Filtraggio dei Fallimenti: Le righe del dataset sono state filtrate per includere solo i casi in cui si è verificato un fallimento (failure = 1). Questo riduce ulteriormente il volume di dati da processare, concentrandosi solo sugli eventi di interesse.
- Ristrutturazione del Campo Data: Il campo data, originariamente nel formato yyyy-MM-dd'T'HH:mm:ss.SSSSSS, è stato convertito nel formato dd-MM-yyyy.
- Raggruppamento e Conteggio: I dati filtrati sono stati raggruppati per giorno (data) e per vault_id. Per ogni gruppo, è stato calcolato il numero di fallimenti. Questa operazione permette di quantificare i fallimenti giornalieri per ciascun vault.
- Filtraggio per Numero di Fallimenti: I gruppi sono stati ulteriormente filtrati per selezionare solo quelli che hanno subito esattamente 4, 3 o 2 fallimenti, identificando i vault con un numero specifico di problemi.
- Riordinamento dei Dati: I risultati sono stati ordinati in base al numero di fallimenti, con i vault che hanno subito più fallimenti in cima alla lista. Questo ordinamento aiuta a dare priorità ai vault più problematici e facilita l'identificazione delle aree critiche.

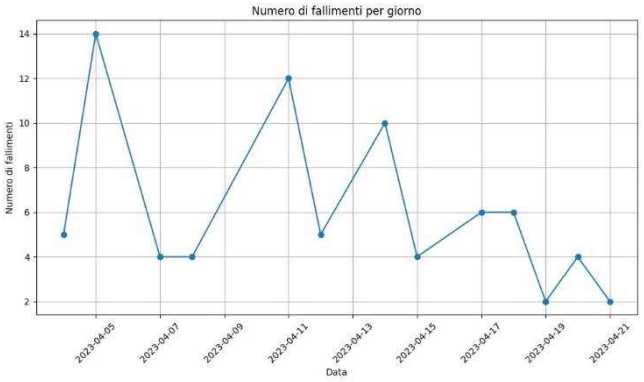

Fig. 4. Fallimenti registrati per giorni

B. Query 2

La seconda query è composta da due fasi distinte. La prima fase mira a classificare i 10 modelli di hard disk che hanno subito il maggior numero di fallimenti fig5, riportando il modello dell'hard disk e il numero totale dei fallimenti per ciascun modello. La seconda fase richiede di calcolare una classifica dei vault che hanno rilevato il maggior numero di fallimenti e, per ogni vault, riportare il numero dei fallimenti e la lista senza ripetizioni dei modelli di hard disk appartenenti al vault e soggetti ad almeno un fallimento fig6. Di seguito Procedura di Implementazione:

Selezione delle Colonne Rilevanti: Le colonne vault_id, model e failure sono state selezionate per ridurre il volume di dati e concentrarsi solo sulle informazioni necessarie per l'analisi.

- Filtraggio dei Fallimenti: Come nella prima query, sono state selezionate solo le righe in cui failure è uguale a 1, per concentrare l'analisi solo sui casi di fallimento degli hard disk.
- Prima fase raggruppamento e Conteggio dei Modelli: I dati sono stati raggruppati per model e per ciascun gruppo è stato calcolato il numero totale di fallimenti. Questo passaggio permette di quantificare i fallimenti per ogni modello di hard disk.
- Ordinamento e Selezione dei Top 10 Modelli: I modelli sono stati ordinati in ordine decrescente in base al numero di fallimenti, e sono stati selezionati i primi 10 modelli con il maggior numero di fallimenti. Questo crea una classifica dei modelli di hard disk più problematici.
- Seconda Fase raggruppamento e Conteggio dei Vault: I dati sono stati raggruppati per vault_id e per ciascun gruppo è stato calcolato il numero totale di fallimenti. Questo passaggio identifica i vault con il maggior numero di fallimenti.
- Raccolta dei Modelli di Hard Disk per Vault: Per ogni vault, è stata creata una lista dei modelli di hard disk che hanno subito almeno un fallimento, rimuovendo le eventuali ripetizioni. Questa lista fornisce una visione chiara dei modelli di hard disk problematici presenti in ogni vault.
- Ordinamento dei Vault: I vault sono stati ordinati in ordine decrescente in base al numero di fallimenti, fornendo una classifica dei vault più problematici.

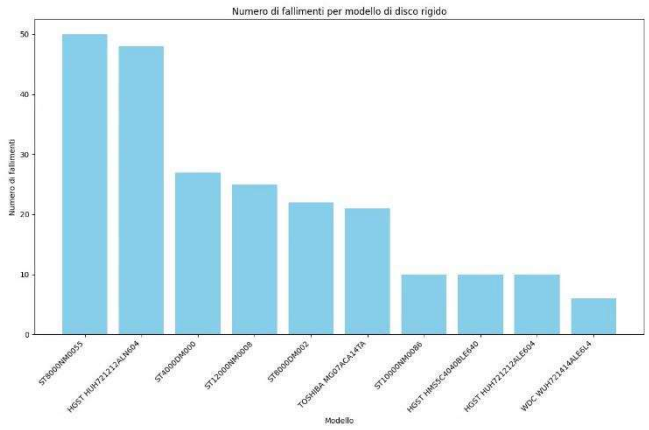

Fig. 5. Modelli che hanno subito piu fallimenti

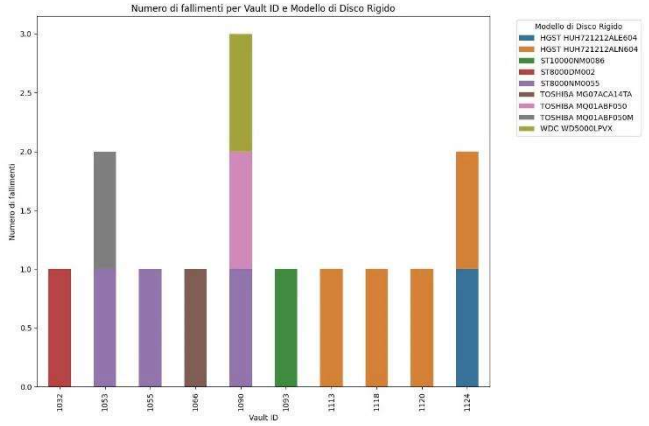

Fig. 6. Vault che hanno registrato più fallimenti

C. *Query 3*

L'obiettivo della terza query è calcolare il minimo, 25-esimo, 50-esimo, 75-esimo percentile e il massimo delle ore di funzionamento degli hard disk, distinguendo tra quelli che hanno subito fallimenti e quelli che non ne hanno subito. Questa analisi statistica è utile per comprendere la distribuzione delle ore di funzionamento e identificare eventuali correlazioni tra il tempo di utilizzo e i fallimenti degli hard disk fig. Procedura di Implementazione:

- Selezione delle Colonne Rilevanti: Sono state selezionate le colonne model, failure e hours_in_operation per concentrarsi esclusivamente sulle informazioni necessarie per l'analisi delle ore di funzionamento in relazione ai fallimenti. Partizionamento degli Hard Disk: Gli hard disk sono stati partizionati in due gruppi utilizzando le API Window di spark.sql, hard disk che hanno subito fallimenti (failure = 1), hard disk che non hanno subito fallimenti (failure = 0).
- Questo è stato realizzato utilizzando le funzioni di aggregazione e finestra di Spark SQL. Le API Window permettono di eseguire calcoli su partizioni di dati, fornendo un potente strumento per analisi statistiche avanzate.
- Calcolo delle Statistiche: Utilizzando le funzioni di aggregazione di Spark, sono stati calcolati i valori minimi, massimi e i percentili (25-esimo, 50-esimo, 75-esimo) delle ore di funzionamento per ciascun gruppo. Le funzioni di finestra sono state impiegate per definire le partizioni e calcolare i percentili con precisione.

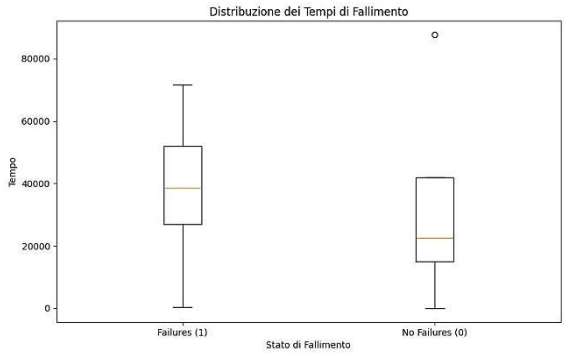

Fig. 7. Distribuzione delle ore di funzionamento

VI. Analisi Prestazioni

Per l'analisi delle prestazioni, sono state testate diverse configurazioni dei nodi di Spark, sia in modo orizzontale (aumentando il numero di nodi) che verticale (aumentando le risorse assegnate a un singolo nodo). L'obiettivo era trovare un compromesso tra le risorse utilizzate e una buona tolleranza ai guasti, garantendo al contempo prestazioni accettabili. Dai risultati degli esperimenti (Tab. 1), è emerso che l'uso del formato Parquet è preferibile per l'analisi dei dati, soprattutto per le query che operano sulle colonne. Questo formato non solo risparmia spazio di archiviazione, ma organizza i blocchi di una stessa colonna in maniera contigua, rendendo l'accesso ai dati più veloce. Analizzando i dati delle diverse configurazioni, la configurazione ottimale risulta essere 1 worker(3 core, 3 GB di memoria). Questo perché il dataset, dopo il preprocessamento, non è troppo grande, e l'overhead di scambio dati e gestione delle risorse tra i vari nodi è ridotto.

Tuttavia, per una maggiore tolleranza ai guasti, è stato deciso di utilizzare 3 nodi (3W-1C-1G), in modo da distribuire meglio il carico e garantire una maggiore resilienza nel caso in cui un nodo worker fallisca.

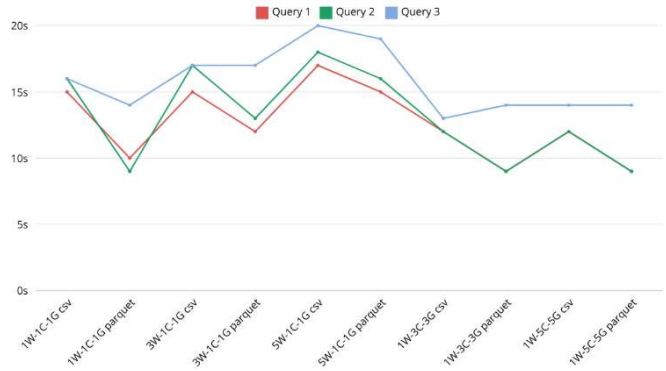

Fig. 8. Tempi totali spark