Batch Processing con Spark

Secondo 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 2 query assegnate nel secondo 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 stream processing Apache Flink per rispondere ad alcune query su dati di telemetria di circa 200k hard disk nei data center gestiti da Backblaze. Essendo i dati già disponibili è necessario simulare la produzione di 23 giorni di monitoraggio per applicare le tecniche di data stream processing.

Il lavoro è suddiviso nelle seguenti fasi principali::

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

Nel progetto si utilizzano i seguenti framework:

- Apache Nifi per la fase di ingestion
- Apache Kafka per la produzione dei dati
- Apache Flink per il processamento dei dati
- Prometheus per il monitoraggio delle prestazioni

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.
- S194:temperature_celsius: rappresenta la temperatura registrata dal disco.

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 volume condiviso con l'host. La scelta di un singolo nodo è giustificata dalla sufficiente capacità di NiFi di gestire il carico di lavoro previsto.

Apache Kafka: Kafka viene utilizzato per registrare i dati provenienti dai producer per poi consegnarli al framework di processamento:

- Kafka-brokers: Si occupano di gestire le partizioni dei dati
- Zookeeper: framework per la coordinazione

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

- Jobmanager: è responsabile dell'accettazione del job da eseguire, coordina ed assegna le risorse, si occupa del monitoraggio dei task.
- Taskmanager: è responsabile dell'esecuzione effettiva dei task, si occupa della comunicazione e della gestione dello stato.

Prometheus: viene impiegato per tenere traccia dei dati di monitoraggio dei taskmanager e jobmanager e fare query per il monitoraggio di latenza e throughtput.

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.

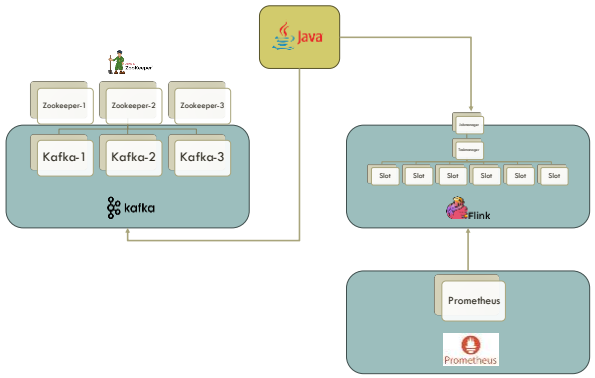

Fig. 1. Architettura

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.

Elaborazione: Alcune righe del dataset avevano una ridondanza degli header, il primo QueryRecord è necessario per rimuoverle, il processor UpdateRecord viene impiegato per la trasformazione del campo date da formato yyyy-mm-ddThh:MM:ss.SSSSSS in timestamp, in quanto durante la fase di processamento verranno usati per l'EventTime delle finestre. L'ultimo QueryRecord divide il dataset in 3 parti applicando l'operatore modulo per rendere la divisione equa.

Salvataggio dei Dati: I dati preprocessati, vengono salvati sul volume condiviso con l'host, sono tre file che servono per rappresentare tre cluter con i vari vault_id.
Diagrammi di Flusso:
Fig 1: Diagramma del flusso di lavoro di NiFi riassumeme tutta la procedura descritta.

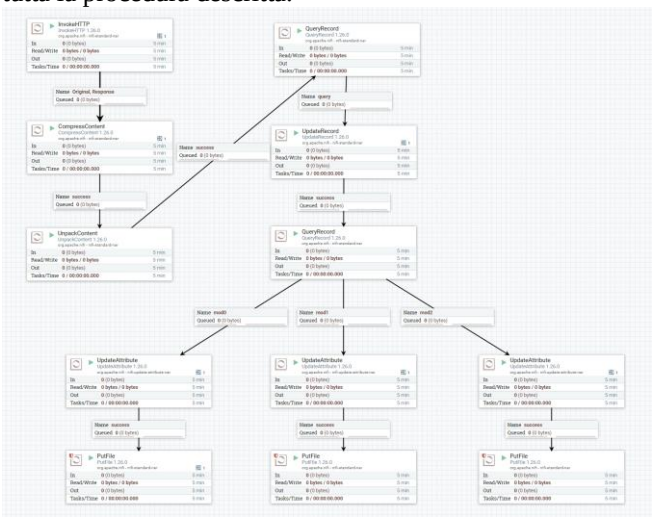

Fig. 2. Nifi-flow preprocessing.

V. Ingestion - Kafka

L'ingestion viene fatto da apache kafka, gruppo di broker-server che gestisce le partizioni, il topic utilizzato dal producer è dsp-flink che verrà consumato dal framework di processamento. In questa fase viene usato un client Kafka scritto in java che cerca di simulare, velocizzando la produzione, 23 giorni di monitoraggio. L'applicazione simula tale scenario impiegando 3 thread che rappresentano 3 cluster, i quali si occupano di immettere i dati in Kafka. Per la simulazione si è scelto di non inviare una tupla per volta, in quanto il delay generato dalla comunicazione avvrebbe allungato di molto i tempi della simulazione, ma vengono pubblicati un gruppo di 500 tuple ogni 240ms, cercando di non appesantire ulteriormente il carico di lavoro della componente di processing.

VI. Processamento - Flink

Il processamento dei dati è stato eseguito utilizzando Apache Flink, sfruttando il linguaggio di programmazione Java versione 11 compatibile con il container ufficiale di Flink. Per scrivere le query, sono state utilizzate le API della libreria flink per i DataStream, per eseguire le operazioni di analisi sui dati di telemetria, ogni task genera un Datastream<Tuple#> per l'operatore successivo.

.

Formato di Input: Le query richiedono a kafka un insieme di tuple, esse hanno tutti i campi che andranno gestiti opportunamente dalla query interessata. Le tuple comprendono tutti i campi del file sorgente. E alcuni valori possono essere mancanti, l'operatore che gestirà la query mappa deve mappare tali tuple.
Salvataggio dei Risultati: I risultati delle query vengono salvati nel volume condiviso con l'host results nelle cartelle

- ./results/query1:/results/queries/query1

- ./results/query2:/results/queries/query2.

Gestione delle Finestre:

Le finestre sono state impostate per avere durata (EventTime) di 1, 3, 23 giorni. Come strategia di Watermark è stato scelto forBoundedOutOfOrderness con ritardo ammesso di 1 giorno, siccome abbiamo più produttori alcune tuple potrebbero arrivare in ritardo.

Output:

Si ha una Map di sink per gli output per generare l'output nel file che indica il nome della query e la durata della finestra scelta.

A. Query 1

Per i vault (campo vault temperature id) con identificativo compreso tra 1000 e 1020, calcolare il numero di eventi, il valor medio e la deviazione standard della temperatura misurata sui suoi hard disk (campo s194 celsius). Si faccia attenzione alla possibile presenza di eventi che non hanno assegnato un valore per il campo relativo alla temperatura. Selezione delle Colonne Rilevanti: Sono state selezionate solo le colonne necessarie per l'analisi: data, vault_id e failure. Calcolare la query sulle finestre temporali:

• 1giorno (event time)

• 3giorni(eventtime);

• dall'inizio del dataset.

Di seguito i passi:

- Map: la tupla in ingresso in Datastream<> i campi di interesse per questa query sono [0]: timestamp, [4]: vault_id e [25]: s194_temperature_celsius. Alcuni valori di temperatura non sono presenti quindi si è scelto di applicare una media mobile per assegnare il valore nel caso il dato fosse mancante.
- Filtraggio: vengono presi i vault interesse compresi dal 1000 al 1020.
- Le finestre vengono calcolate usando il metodo process, la classe che implementa tale funzione e TemperatureStatsMetrics che applica l'algoritmo di Welford. È una KeyWindow dove per ogni vault vengono calcolati i dati di interesse.
- L'output ha necessità di essere mappato per poter essere passato al sink nella forma corretta.

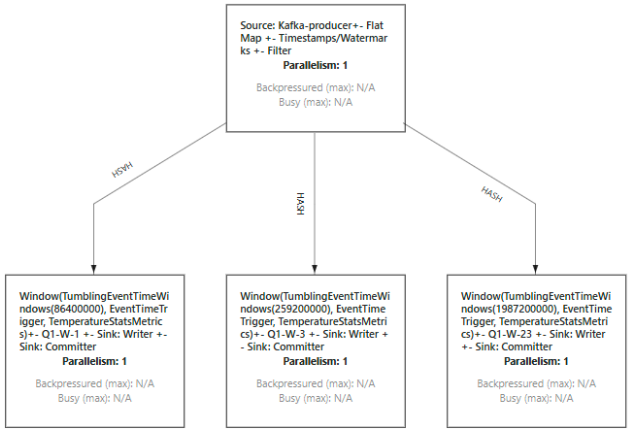

Fig. 3. DAG jobmanager:8081 query 1 parallelism=1

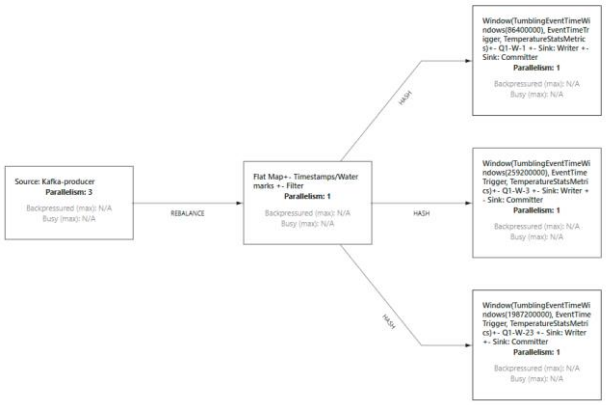

Fig. 4. DAG jobmanager:8081 Query1 parallelism=3

B. Query 2

Calcolare la classifica aggiornata in tempo reale dei 10 vault che registrano il piu alto numero di fallimenti nella stessa giornata. Per ogni vault, riportare il numero di fallimenti ed il modello e numero seriale degli hard disk guasti. Calcolare la query sulle finestre temporali:

• 1 giorno (event time) • 3 giorni (event time); • dall'inizio del dataset.

- 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.
- Map: questa query necessità dei campi [0]: timestamp, [1]: serial_number, [2]: model, [3]: failure, [4]: vault_id.
- Filtraggio: c'è necessità di selezionare solo le tuple il cui campo failure sia 1.
- KeyStream: per ogni chiave (vault_id) bisogna contare quanti elementi sono presenti, simile al word count, operazione gestita dalla classe KeyCountFailures.

- WindowAll: Ora che si hanno le tuple bisogna fare la classifica, questa funzionalità viene presa in carico dalla classe GroupFailures, che usa l'algoritmo di minHeap per stilare la classifica dei 10 vault che hanno registrato il maggior numero di fallimenti.
- Aggiustamento dell'output: è stato usato una map per i modelli con i relativi serial number quindi i dati subiscono un'ulteriore trasformazione.

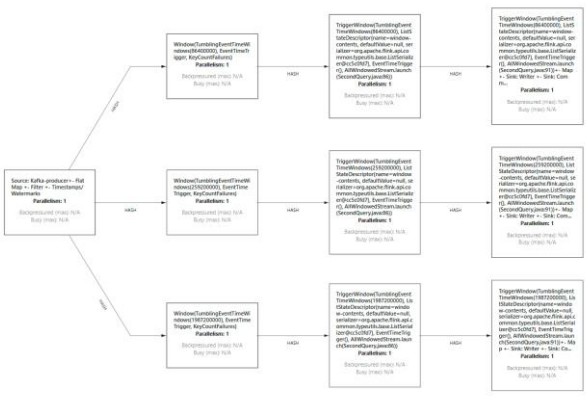

Fig. 5. DAG jobmanager:8081 Query2 parallelism=1

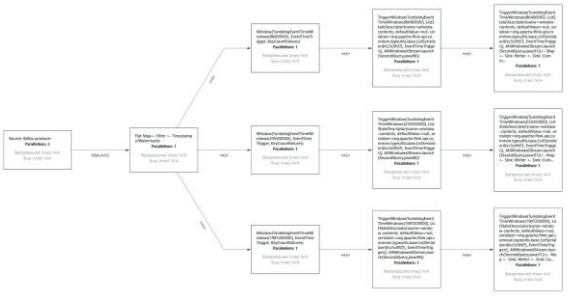

Fig. 6. DAG jobmanager:8081 Query2 parallelism=3

VII. Analisi prestazioni

Per l'analisi delle prestazioni, sono state le due query in contemporanea e le query prese singolarmente con una configurazione di jobmanager, 1 taskmanager e parallelismo. Per la latenza sono state usate le metriche di flink che vengono esposte su prometheus gli operatori sono presi secondo l'ID dato dall'hash. Per il throughput si è usato la una metrica custom. e. La metrica è stata calcolata prendendo prima di tutto il tempo di creazione dell'operatore; quando una tupla passa per l'operatore, viene preso l'istante attuale, si calcola la differenza con l'istante di creazione dell'operatore, si incrementa il numero di tuple passate fino a quel momento per l'operatore e si dividono i due valori: quello che otteniamo quindi è una stima dell'evoluzione del throughput nel tempo. Si può notare che i tempi di latenza aumentano leggermente quando vengono bisogna lavorare con più query contemporaneamente, ed è maggiore nel caso della query 2 in quanto c'è più scambio di messaggi tra gli operatori della finestra. Mentre per quanto riguarda il throughtput cambia poco. è maggiore nelle finestre più piccole perché lavora con più dati, infatti il throughput della query 1 risulta essere più grande rispetto la query 2. Per analisi future è possibile analizzare la latenza e throughput aumentando il grado di parallelismo aspettandoci un aumento della latenza ma un leggero miglioramento di throughput, in questo progetto non è stato possibile a causa di un problema con i sink nel caso di run con parallelize > 1.

Fig. 7. Query1 & 2 Latency

Fig. 8. Query1 & 2 Throughtput

Fig. 9. Query1 Latency

Fig. 10. Query1 Throughput

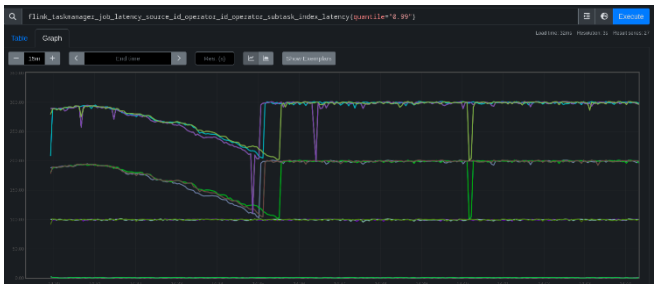

Fig. 11. Query2 Latency

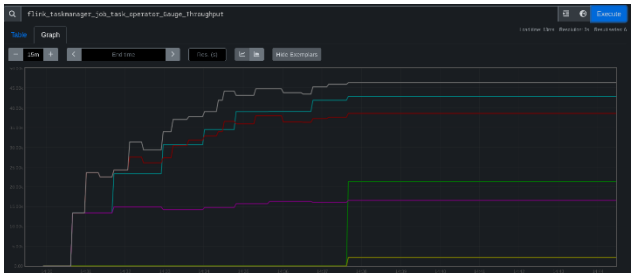

Fig. 12. Query2 Throughput