Relazione progetto

Andrea Cantarini
CdS di Ingegneria Informatica
Università degli Studi di Roma "Tor Vergata"
Rome, Italy
andrea.cantarini@alumni.uniroma2.eu

***Abstract*—La presente relazione è atta a descrivere un'architettura distribuita in grado di eseguire interrogazioni su un dataset di grandi dimensioni.**

***Index Terms*—Apache Spark, HDFS, Docker, Big Data**

I. Istruzioni

Lo scopo del progetto è 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 [1]. Per gli scopi di questo progetto, viene fornita una versione ridotta del dataset indicato nel Grand Challenge della conferenza ACM DEBS 2024.
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 front dei 5 milioni del dataset originario). La Tabella 1 descrive i campi di ogni evento. Gli attributi rilevanti per questo progetto sono evidenziati nella tabella.

A. Query

Si richiede di rispondere alle seguenti query:

1) Per ogni giorno, per ogni vault (si faccia riferimento al campo vault id), calcolare il numero totale di fallimenti. Determinare la lista di vault che hanno subito esattamente 4, 3 e 2 fallimenti.

   Esempio di output:
   # DD-MM-YYYY, vault id, failures count
   11-04-2023, 1090, 4
   ...
   19-04-2023, 1120, 3
   ...
   01-04-2023, 1055, 2
   ...
2) Calcolare la classifica dei 10 modelli di hard disk che hanno subito il maggior numero di fallimenti. La classifica deve riportare il modello di hard disk e il numero totale di fallimenti subiti dagli hard disk di quello specifico modello.
   In seguito, calcolare una seconda classifica dei 10 vault che hanno registrato il maggior numero di fallimenti. Per ogni vault, riportare il numero di fallimenti e la lista (senza ripetizioni) di modelli di hark disk soggetti ad almeno un fallimento.

   Esempio di output (i valori indicati sono solo a titolo esemplificativo):
   # model, failures count
   HGST HUH721212ALN604, 47
   ST8000NM0055, 45
   ST4000DM000, 28
   ...

   # vault id, failures count, list of models
   1113, 16, HGST HUH721212ALN604
   1093, 9, ST10000NM0086
   1090, 9, ST8000NM0055, TOSHIBA MQ01ABF050, WDC WD5000LPVX
   ...

B. Consegna

Il risultato di ciascuna query deve essere consegnato in formato CSV. Inoltre, si chiede di valutare sperimentalmente i tempi di processamento delle query sulla piattaforma di riferimento usata per la realizzazione del progetto e di riportare tali tempi nella relazione e nella presentazione del progetto. Tale piattaforma può essere un nodo standalone, oppure è possibile utilizzare un servizio cloud per il processamento di Big Data (e.g., Amazon EMR) avvalendosi del grant a disposizione.

II. Architettura

A. HDFS

I dati su cui eseguire le interrogazioni sono caricati su un cluster HDFS composto da un master e tre slave. Ognuno di questi viene eseguito su un apposito container Docker.
Il nodo master espone le porte 9870 e 54310, mappate alle stesse porte sulla macchina host e utilizzate rispettivamente per accedere all'interfaccia web e allo storage distribuito dall'interno dell'applicazione mentre questa è in esecuzione.

B. Spark

L'esecuzione delle query avviene all'interno di un cluster Apache Spark composto da un master, un worker e un history server. Come per HDFS, anche queste componenti sono tutte eseguite all'interno di container Docker appositi.
Il nodo master espone le porte 8080, 7070 e 4040, per

[1] sha1sum di raw data medium-utv sorted.csv: f5667bf30be58fbf83016f83924b29e65e6246f3; sha1sum del file compresso in TAR GZ: b7d026fdf9b2d14f57400fe206dee9c6f87c7e59

accedere con la prima all'interfaccia web e con la seconda all'applicazione. L'ultima porta non è stata utilizzata.

C. Docker

Al fine di garantire la comunicazione, i container HDFS e Spark sono posti all'interno della stessa network Docker, denominata `sabd-network`.

III. Applicazione

A. Tecnologie utilizzate

L'esecuzione delle query avviene tramite un'applicazione Java in esecuzione sul cluster di Apache Spark descritto precedentemente. Si sfrutta la libreria Java Spark messa a disposizione dal framework stesso per interfacciarsi con il cluster, definire ed eseguire le query.
Tra le varie tecnologie messe a disposizione da Spark, in questo contesto verranno utilizzati i Resilient Distributed Dataset, a cui si farà riferimento come RDD da ora in poi.

B. Data ingestion

Dopo aver caricato il dataset in HDFS come /data.csv, è possibile andarne a leggere il contenuto tramite la libreria di Java Spark. In particolare, si sfrutta l'oggetto SparkSession, inizializzato con parametri di configurazione quali URL del nodo master e nome dell'applicazione, per leggere il contenuto del file CSV e ottenere un oggetto JavaRDD¡String¿. A seguito di ciò, poiché non tutte le colonne del dataset sono interessanti ai fini del presente progetto, si utilizza un parser CSV che recuperi solo le colonne di interesse, scartando le altre. Il risultato è un JavaRDD con cinque colonne:

- timestamp (String)
- numero seriale dell'hard-disk (String)
- modello dell'hard-disk (String)
- id del vault in cui l'hard-disk è presente (Long)
- fallimento (valore booleano trattato come Long)

Avendo ora i dati caricati e a disposizione, è possibile definire ed eseguire le query.

C. Definizione delle query

1) Query 1: La prima query è composta dalla seguenti fasi:

1) filtraggio per eliminare i record che non costituiscono eventi di fallimenti
2) mappatura di ogni riga in coppie ((timestamp, vault_id), 1)
3) somma di tutti gli elementi in base alla chiave
4) ulteriore mappatura dei risultati per ottenere una terna (timestamp, vault_id, failures_counts)
5) ordinamento dei dati in base alla prima colonna, ossia al timestamp

Infine, viene eseguita un'azione di collect per eseguire effettivamente la query sul cluster Spark e ottenere i risultati.

2) Query 2 - 1: La prima delle seconde query è composta dalla seguenti fasi:

1) filtraggio per eliminare i record che non costituiscono eventi di fallimenti
2) mappatura di ogni riga in coppie (model, 1)
3) somma di tutti gli elementi in base alla chiave
4) scambio di posizione delle colonne
5) ordinamento dei risultati in base alla prima colonna

Infine, viene eseguita un'azione che consente di prendere le prima dieci entry del risultato.

3) Query 2 - 2: La seconda delle seconde query è composta dalla seguenti fasi:

1) filtraggio per eliminare i record che non costituiscono eventi di fallimenti
2) mappatura di ogni riga in coppie (vault_id, (1, model))
3) operazione di reduce in base alla chiave che esegue la somma degli elementi in prima posizione nel value, ossia il flag di fallimento, mentre esegue la concatenazione - senza ripetizioni - degli elementi in seconda posizione, ossia dei modello di hard-disk.

Infine, come nella precedente query, viene eseguita un'azione che consente di prendere le prima dieci entry del risultato.

D. Scrittura dei risultati

I rusultati sono copiati all'interno di HDFS. Nello specifico, con i risultati ottenuti da ogni query viene creato un oggetto Dataset dal quale è possibile eseguire la scrittura su HDFS in maniera più semplice. Soltanto nel caso della prima query viene eseguito un processamento intermedio per convertire il formato dei timestamp in DD-MM-YYYY.
I risultati sono riportati, in formato CSV, all'interno della cartella /Results.

IV. Automatizzazione

Sono messi a disposizione nel progetto degli script shell che consentono di automatizzare il deployment, il caricamento dei dati e l'esecuzione delel query. Gli script sono posti all'interno della cartella /Scripts.

A. Avvio e spegnimento dell'architettura

Per avvia i container che compongono l'architettura dell'applicazione, eseguire lo script `run.sh`. Per spegnere l'architettura, eseguire `stop.sh`.

B. Avvio e spegnimento di HDFS

Le immagini HDFS richiedono un avvio manuale del master e dei worker. Per l'avvio, eseguire lo script `start-hdfs.sh`. Tale script fa uso del comando `docker compose exec` per eseguire dei comandi direttamente all'interno dei container. Per fermare HDFS, eseguire stop-hdfs.sh.

C. Caricamento dei dati

È possibile caricare il dataset su HDFS eseguendo lo script `load-data.sh`, passando come argomento il path al dataset da caricare.

D. Build ed esecuzione dell'applicazione

Per eseguire la build Maven del progetto, eseguire `make.sh`. Per avviare l'applicazione, eseguire `deploy.sh`. Questi ultimi due script si trovano nella directory root del progetto.

V. Miglioramenti

È possibile apportare dei miglioramenti sia all'architettura, sia al codice dell'applicazione.
Si può inserire un layer di data ingestion tra HDFS e Spark, ad esempio tramire Apache NiFi, in modo da eseguire l'applicazione con tutti i dati già a disposizione e nel formato corretto, senza dover scartare manualmente delle colonne.
Al fine di migliorare le prestazione dell'esecuzione delle query, nonché la leggibilità del codice, si può sfruttare la libreria Spark SQL per Java. Quest'ultima, oltre ad avere prestazioni migliori rispetto agli RDD, consente di scrivere le query in linguaggio SQL, rendendo così il codice più leggibile e dunque manutenibile.