Sistemi e Architetture per Big Data - AA 2020/2021

Secondo progetto

Giuseppe Lasco
Dipartimento di Ingegneria dell'Informazione
Università degli studi di Roma "Tor Vergata"
Roma, Italia
giuseppe.lasco17@gmail.com

Marco Marcucci
Dipartimento di Ingegneria dell'Informazione
Università degli studi di Roma "Tor Vergata"
Roma, Italia
marco.marcucci96@gmail.com

***Abstract*—Questo documento riporta i dettagli implementativi riguardanti l'analisi mediante *Flink* del dataset relativo a dati provenienti da dispositivi Automatic Identification System (AIS) contenenti informazioni riguardo lo stato di navi in movimento per garantire la sicurezza di quest'ultime in mare e nei porti. Viene, inoltre, descritta l'architettura a supporto dell'analisi e gli ulteriori *framework* utilizzati.**

I. Introduzione

L'analisi effettuata si pone lo scopo di rispondere a delle query relative a classifiche e statistiche riguardanti le navi e le tratte presenti nel dataset.

Dataset

Il dataset preso in considerazione è *prj2_dataset.csv*, il quale contiene dati rigurdanti gli identificativi e le caratteristiche istantanee delle navi e delle tratte. I campi di interesse sono:

- **ID**: stringa esadecimale che rappresenta lidentificativo della nave;
- **SHIP TYPE**: numero intero che rappresenta la tipologia della nave
- **LON**: numero in virgola mobile che rappresenta la coordinata cartesiana in gradi decimali della longitudine data dal GPS;
- **LAT**: numero in virgola mobile che rappresenta la coordinata cartesiana in gradi decimali della latitudine data dal sistema GPS;
- **TIMESTAMP**: rappresenta listante temporale della segnalazione dellevento AIS; il timestamp espresso con il formato GG-MM-YY hh:mm:ss (giorno, mese, anno, ore, minuti e secondi dellevento);
- **TRIP ID**: stringa alfanumerica che rappresenta lidentificativo del viaggio; composta dai primi 7 caratteri (inclusi 0x) di SHIP ID, concatenati con la data di partenza e di arrivo.

La frequenza di produzione di tali dati in funzione dello stato di moto, con un periodo temporale variabile tra i 2 secondi in fase di manovra a 5 minuti in fase di navigazione ad alta velocit. Inoltre, l'area marittima limitata alla zona del Mar Mediterraneo descritta dalle seguente coordinate: LON $\in$ [-6.0, 37.0] LAT $\in$ [32.0, 45.0]. Tale area stata suddivisa in celle rettangolari di uguale dimensione; i settori di LAT vengono identificati dalle lettere che vanno da A a J, mentre i settori di LON dai numeri interi che vanno da 1 a 40. Ad ogni cella associato un *id* dato dalla combinazione della lettera del settore LAT e dal numero di settore LON.

Query

L'obiettivo di questo progetto è quello di implementare ed eseguire tre query utilizzando *Flink*.

La prima query ha come scopo quello di calcolare, per il Mar Mediterraneo Occidentale, il numero medio giornaliero di navi militari, navi per trasporto passeggeri, navi cargo e le restanti tipologie, utilizzando finestre temporali di tipo *Tumbling* da 7 giorni e da 1 mese.

La seconda query consiste nel determinare le prime 3 celle per le quali il grado di frequentazione è più alto, nelle due fascie orarie 00:00-11:59 e 12:00-23:59, Mar Mediterraneo Occidentale ed Orientale. il grado di frequentazione di una cella viene calcolato come il numero di navi diverse che attraversano la cella nella fascia oraria in esame. Sono state utilizzate finestre temporali a 7 giorni e 1 mese.

L'ultima query consiste nel determinare le prime 5 tratte per cui la distanza percorsa fino a quel momento è più alta. Per il calcolo della distanza stata considerata la distanza euclidea.

Framework

Come *framework* di processamento stream è stato utilizzato *Apache Flink* che riceve i dati dal sistema di messaging *Apache Kafka*.

II. Architettura

L'architettura si compone di due container *Docker*, su cui eseguono i servizi di *Apache Zookeeper* e *Apache Kafka* che comunicano tra di loro attraverso la stessa rete, creata appositamente dal servizio *Docker Compose*. Inoltre, sempre sulla stessa macchina, una JVM ospita l'esecuzione di *Apache Flink* e un processo *Java* immette i dati nel sistema di Publish/Subscribe.

Producer

Il Producer si occupa di leggere il dataset, mantenuto in un file CSV locale, e di pubblicarne il contenuto presso il topiche di Kafka (query) da cui *Flink* recupera i dati. La scrittura

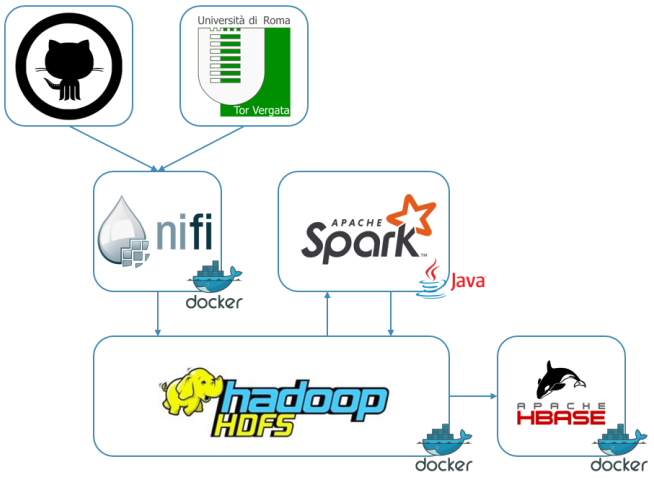

Fig. 1: Schema dell'architettura

sulla topica viene effettuata riga per riga a intervalli variabili, proporzionali ai timestamp reali presenti nel dataset, al fine di simulare una vera sorgente di dati real time accelerata. L'ordinamento del dataset è stato effettuato utilizzando una struttura dati che permette di ordinare all'inserimento dei record, ovvero la *TreeMap*. La gestione di chiavi uguali (timestamp) è stata effettuata utilizzando come valore della coppia key-value della *TreeMap*, una lista, popolata dai record da inviare, in modo da non perdere occorrenze. Per garantire la corretta esecuzione del processamento su *Flink* in base all' event time, è stato necessario estrarre la data di occorrenza dell'evento di ogni riga e impostarla come timestamp della relativa tupla alla pubblicazione sulla topica.

Apache Kafka

Kafka il sistema di messaggistica di tipo publish-subscribe utilizzato per l'ingestion di dati nei sistemi di processamento e per l'export dei risultati. Il cluster, realizzato con *Docker Compose*, prevede un container con *Zookeeper*, necessario per la coordinazione, e altri tre container con la funzione di *Kafka broker*. Sono state create 7 *topiche*: una per le tuple in input a Flink, due per l'output della prima query (settimanale e mensile), due per l'output della seconda query (settimanale e mensile) e altrettante per quello della terza query (per un'ora e per due ore). Per incrementare la tolleranza ai guasti, ogni *Kafka topic* impostata per avere un grado di replicazione pari a 2 (una replica *leader* ed una replica *follower*) e, allo stesso tempo, una sola partizione. La scelta della singola partizione dovuta alla necessit di mantenere le tuple ordinate allinterno del sistema di messaggistica; in *Kafka*, infatti, la garanzie di ordinamento sono valide soltanto nellambito di una singola partizione.

Apache Flink

Flink è il framework di *data stream processing* utilizzato per l'esecuzione delle tre query precedentemente descritte. I dati necessari al processamento sono presi direttamente dalla *topica* query in *Kafka*. L'*event time* viene determinato in automatico da *Flink* recuperandolo dal campo *timestamp* del record *Kafka*. Il flusso così ottenuto rappresenta lo stream che le query devono manipolare al fine di calcolare le statistiche richieste. Al fine di preprocessare i dati ed eseguire le *query*, viene utilizzato *Apache Flink* in locale, tramite lo script `$FLINK_HOME/bin/flink run`. Di seguito viene mostrata la topologia che sar successivamente descritta pi nel dettaglio.

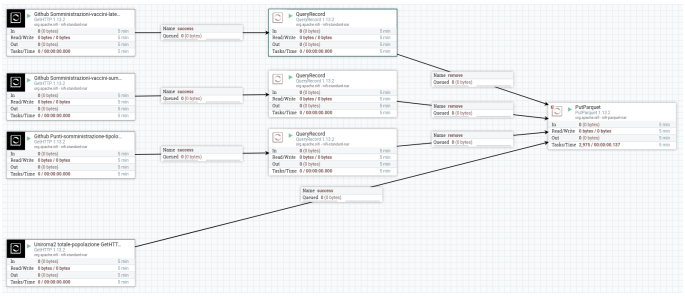

Fig. 2: NiFi template

III. QUERY

Al fine di effettuare le query è stato necessario uno step di preprocessamento. Lo stream in ingresso viene generato utilizzando un consumer che riceve i dati dalla topica *query*. I *record* vengono trasformati in oggetti di tipo *ShipData* contenenti le informazioni necessarie al processamento delle richieste; in particolare viene definita la cella e la sezione del Mar Mediterraneo (Occidentale o Orientale) di appartenenza a partire dalle coordinate. Una operazione di filtraggio assicura che i dati rientrino nella porzione di mare indicata dalla traccia. Lo stream di oggetti *ShipData* in uscita viene, in seguito, dato in ingresso alle topologie di processamento delle query.

Query 1

Per quanto riguarda la prima query, il processamento inizia filtrando i record, in modo da prenderer in considerazione solo quelli relativi al Mar Mediterraneo Occidentale. Il processamento tramite *Flink* prosegue sfruttando le finestre temporali di tipo *tumbling* settimanali e mensili, parallelizzando attraverso il *keyBy* per cella. Su tutte le finestre viene applicata una funzione di *aggregate* personalizzata che permette, al contrario della *process*, di aggiornare le statistiche della *window* ogni volta che una tupla le viene assegnata; questa accortezza consente di evitare picchi di carico dovuti alla computazione in blocco di tutte le tuple assegnate a una finestra al completamento della stessa. In particolare, la funzione di aggregate permette di contare il numero di navi di un certo tipo passanti per una determinata cella nel periodo della finestra. Infine, attraverso una *ProcessWindowFunction* user defined, si è prelevato il timestamp relativo all'inizio della finestra. Una *map* ha permesso di formattare correttamente i dati da passare al *sink*.

Query 2

Per quanto riguarda la seconda query, il processamento tramite *Flink* inizia sfruttando le finestre temporali di tipo *tumbling* settimanali e mensili, parallelizzando attraverso il *keyBy* per cella. Su tutte le finestre viene applicata una funzione di *aggregate* personalizzata che permette di contare il numero di navi distinte passanti per una determinata cella nel periodo della finestra considerando le fasce orarie 00:00-11:59, 12:00-23:59 e Mar Mediterraneo Occidentale ed Orientale. Infine, attraverso una *ProcessWindowFunction* user defined, si è prelevato il timestamp relativo all'inizio della finestra. Una finestra di tipo *WindowAll* è stata necessaria al fine di generare la classifica delle celle più frequentate relative ad una certa zona di mare e una determinata fascia oraria. Una *map* ha permesso di formattare correttamente i dati da passare al *sink*.

Query 3

Per quanto riguarda l'ultima query, il processamento tramite *Flink* inizia sfruttando le finestre temporali di tipo *tumbling* di una e due ore, parallelizzando attraverso il *keyBy* per tripId. Su tutte le finestre viene applicata una funzione di *aggregate* personalizzata che permette calcolare la distanza percorsa dalla nave in questione, tenendo traccia delle coordinate relative al record precedende, in modo da poter calcolare la distanza euclidea tra i punti e sommarla alla distanza cumulata fino a quel momento. Infine, attraverso una *ProcessWindowFunction* user defined, si è prelevato il timestamp relativo all'inizio della finestra. Una finestra di tipo *WindowAll* è stata necessaria al fine di generare la classifica dei viaggi con distanza coperta maggiore nel tempo della finestra. Una *map* ha permesso di formattare correttamente i dati da passare al *sink*.

IV. BENCHMARK

L'esecuzione del progetto e la valutazione delle prestazioni sono state eseguite su *Linux Mint 20.1 Cinnammon*, *CPU AMD Ryzen 5 3600*, 6 core, 12 thread e 16 GB di RAM, con archiviazione su *SSD*.

TABLE I: Latenze e throughput

Query	Throughput (tuple/sec)	Latenza (sec/tupla)
Query 1 weekly	330.751	0.003
Query 1 monthly	381.912	0.003
Query 2 weekly	320.114	0.003
Query 2 monthly	344.525	0.003
Query 3 one hour	330.472	0.003
Query 3 two hour	344.525	0.003

*Le metriche si riferiscono alla durata complessiva del replay di 10 secondi

Per poter eseguire una valutazione sperimentale dei benchmark di ogni operatore, è stata utilizzata, per ognuno di essi, una struttura tenente conto sia del numero di tuple processate che del tempo impiegato; l'accesso a tale struttura è effettuato, per evitare inconsistenze nel caso di aggiornamenti simultanei da parte di operatori replicati, mediante lutilizzo di metodi `synchronized`. In tabella I sono riportati i throughput e le latenze medie di processamento dei singoli eventi che attraversano la topologia per le tre query.

REFERENCES

[1] https://spark.apache.org/docs/latest/
[2] https://stackoverflow.com/
[3] https://nifi.apache.org/
[4] https://hadoop.apache.org/docs/stable/
[5] http://spark.apache.org/docs/latest/ml-guide.html
[6] https://commons.apache.org/proper/commons-math/javadocs/api-3.3/org/apache/commons/math3/stat/regression/SimpleRegression.html