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

***Sommario*—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 l'identificativo 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 l'istante temporale della segnalazione dell'evento AIS; il timestamp è espresso con il formato GG-MM-YY hh:mm:ss (giorno, mese, anno, ore, minuti e secondi dell'evento);
- **TRIP ID**: stringa alfanumerica che rappresenta l'identificativo del viaggio; è 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 è 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 velocità. Inoltre, l'area marittima è limitata alla zona del Mar Mediterraneo descritta dalle seguente coordinate: LON $\in$ [-6.0, 37.0] LAT $\in$ [32.0, 45.0]. Tale area è 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 è 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) e per le due aree marittime (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 è stata considerata la distanza di Vincenty.

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 quattro 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*, un processo *Java* immette i dati nel sistema di Publish/Subscribe e un Consumer, costituito anch'esso da un processo *Java*, recupera i risultati delle query dalle topiche. È possibile osservare la struttura dell'architettura in figura 1.

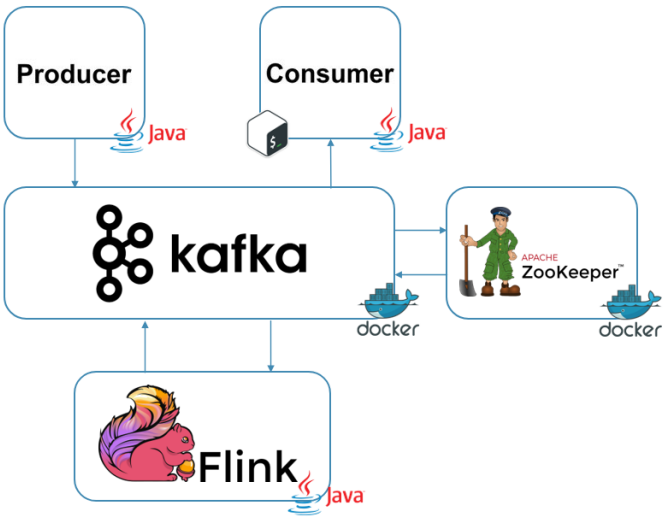

Figura 1: Schema dell'architettura

Producer

Il Producer si occupa di leggere il dataset, mantenuto in un file CSV locale, e di pubblicarne il contenuto presso la topica di Kafka (query) da cui *Flink* recupera i dati. La scrittura 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 *TreeMap*, che permette di ordinare i record ad ogni nuovo valore inserito, considerando come chiave il timestamp. La gestione di chiavi uguali 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 è il sistema di messaggistica di tipo publish-subscribe utilizzato per l'ingestion di dati nel sistema 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* è 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 è dovuta alla necessità di mantenere le tuple ordinate all'interno del sistema di messaggistica; in *Kafka*, infatti, la garanzie di ordinamento sono valide soltanto nell'ambito 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 allo scopo 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`. Alla fine del processamento i risultati vengono immessi nuovamente in *Kafka*, sulle topiche sopra citate. Il parallelismo dei task è stato settato a 12, sfruttando gli hyperthread della macchina. In figura 2 viene mostrata la topologia che sarà successivamente descritta più nel dettaglio.

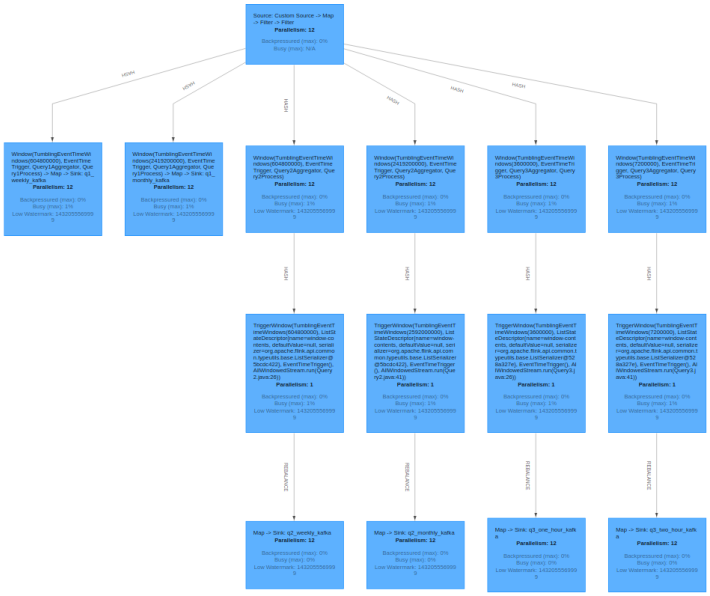

Figura 2: Topologia Flink

Consumer

Il Consumer è rappresentato da un processo *Java* che genera un numero di thread pari al numero di topiche di uscita del sistema. Tali thread, una volta associati alla topica, recuperano i dati e li scrivono su file in formato CSV, dinamicamente, inserendo l'header. La semantica utilizzata è exactly once.

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 prendere 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 di aggiornare dinamicamente la distanza percorsa dalla nave in questione, tenendo traccia delle coordinate relative al record precedente, in modo da poter calcolare la distanza di Vincenty 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 per distanza, nel tempo della finestra. Tale classifica viene stilata dinamicamente utilizzando una funzione di *aggregate* personalizzata. Una *map* ha permesso, infine, 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*.

Tabella I: Latenze e throughput

Query	Throughput (tuple/sec)	Latenza (sec/tupla)
Query 1 weekly	41.598	0.024
Query 1 monthly	29.987	0.033
Query 2 weekly	2.0	0.501
Query 2 monthly	0.438	2.281
Query 3 one hour	206.567	0.005
Query 3 two hour	104.713	0.009

*Le metriche si riferiscono alla durata complessiva del replay di 5 secondi

Per poter eseguire una valutazione sperimentale dei benchmark è stata utilizzata 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 l'utilizzo 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. Come si può notare il throughput delle query cresce al diminuire della dimensione della finestra, in linea con quanto ci si aspetti.

Riferimenti bibliografici

[1] https://ci.apache.org/projects/flink/flink-docs-release-1.13/
[2] https://kafka.apache.org/documentation/
[3] https://lucene.apache.org/core/5_5_2/spatial/org/apache/lucene/spatial/util/GeoDistanceUtils.html
[4] https://stackoverflow.com/