Progetto 2 [SABD]

DAMIANO NARDI, TorVergata, Corso Di Informatica

ACM Reference Format:
Damiano Nardi. 2020. Progetto 2 [SABD]. 1, 1 (July 2020), 5 pages.

1 INTRODUZIONE

In questa relazione si descriverà il lavoro svolto che consiste nella realizzazione delle query(1,2)

2 PROCESSAMENTO DELLE QUERY

Per processare le query 1 e 2 è stato usato il framework apache flink usando le api java, inoltre per confrontare flink con kafka streams è stata fatta la anche query 1 usando kafka streams. Si è utilizzata una macchina virtuale con Ubuntu 20.04 con allocatevi 4 core della cpu (i7-8750H) e 8gb di ram (ddr4 2400 MHz).

2.1 Stremming del dataset

Il dataset viene letto e inviato riga per riga a una topica kafka che viene poi consumata da apache flink o da kafka streams.

2.2 Query1

> Calcolare il ritardo medio degli autobus per quartiere nelle ultime 24 ore (di event time), 7 giorni (di event time) e 1 mese (di event time). L'output della query ha il seguente schema: ts, boro-x, avg-x, ..., boro-z, avg-z

Lo stream del dataset viene letto da una topica kafka, la prima operazione fatta è una

2.2.1 flatMap. In questa flatMap viene processata la singola ringa del dataset andando a mettere come chiave il campo Boro (il quartiere) e come valore un oggetto con questi campi: OccurredOn (data del ritardo), HowLongelayed (ritardo in minuti) e altri campi meno importanti; le righe del dataset che non hanno il campo HowLongDelayed o lo hanno in un formato non previsto vengono ignorate.

2.2.2 EventTime e TumblingEventTimeWindows. A questo punto flink estrae l'event time dai dati processati dalla flatMap e subito dopo crea una TumblingWindows della durata preimpostata tra 1,7,30 giorni

2.2.3 Reduce. Facciamo una reduce (abbiamo che chiave Boro), essenzialmente andiamo a sommare i ritardi di un quartiere (nella stessa finestra) teniamo anche in considerazione la "somma" delle reduce (per la media) fatte e la data più vecchia in ogni reduce.

2.2.4 Map. In questa Map andiamo a calcolare la media dei ritardi sommati per ogni quartiere nella precedente reduce

2.2.5 TumblingEventTimeWindows. specifichiamo ancora la finestra temporale

Author's address: Damiano Nardi, damiano6276@gmail.com, TorVergata, Corso Di Informatica.

2.2.6 Apply. In fine vengono raggruppati tutti i dati nella stessa finestra temporale e viene formattato l'output come richiesto dalle specifiche.

2.2.7 addSink. Infine viene aggiunto un sink che manda lo stream processato a una topica kafka

2.3 Query 1 kafka streams

La query 1 è stata fatta anche su kafka streams le operazioni fatte sono più o meno le stesse, cambia la sintassi, quindi per evitare ridondanza non verranno riportate.

2.4 Query2

> Fornire la classifica delle tre cause di disservizio piu frequenti (ad esempio, Heavy Traffic, Mechanical ‘ Problem, Flat Tire) nelle due fasce orarie di servizio 5:00-11:59 e 12:00-19:00. Le tre cause sono ordinate dalla piu frequente alla meno frequente. L'output della query ha il seguente schema: ts, slot-a, rank-a, slot-p, rank-p .

Lo stream del dataset viene letto da una topica kafka la prima operazione fatta è una

2.4.1 flatMap. In questa flatMap viene processata la singola riga del dataset andando a mettere come chiave il campo Reason con il prefisso "fascia5-11" o "fascia12-19" scelto in base al campo OccuredOn mentre come valore un oggetto con i seguenti attributi OccurredOn,Reason,rank(intero inizialmente pari a 1), list(inizialmente vuota).

2.4.2 EventTime e TumblingEventTimeWindows. A questo punto flink estrae l'event time dai dati processati della flatMap e subito dopo crea una TumblingWindows della durata preimpostata tra 1 o 7 giorni

2.4.3 Reduce. In questa reduce vengono sommati i campi rank e viene presa la data (OccurredOn) minima tra le varie operazioni di reduce

2.4.4 Map. Viene messa la tupla (Reason,rank) dentro la lista e mettiamo come chiave "fascia5-11" o "fascia12-19"

2.4.5 TumblingEventTimeWindows. specifichiamo ancora la finestra temporale

2.4.6 Reduce. In questa reduce viene fatto il l'unione del campo lista (che avevamo iniziato a popolare nella map precedente) per le due fascie "fascia5-11", "fascia12-19"

2.4.7 Map. Adesso prendiamo la dal campo list le 3 Reason che hanno il rank più alto

2.4.8 TumblingEventTimeWindows. Specifichiamo ancora la finestra temporale

2.4.9 Reduce e Map. In fine facciamo una Reduce e una Map al fine di formattare l'output come richiesto dalle specifiche

2.4.10 addSink. Viene aggiunto un sink che manda lo stream processato a una topica kafka

3 TEMPI

Sono stati misurati i tempi di latenza e throughput di messaggi al secondo, In questo paragrafo descriveremo come sono stati calcolati

3.1 Producer

Come accennato prima abbiamo un producer che legge il dataset riga per riga e li passa a una topica kafka durante questa operazione appena prima di inserire la riga sulla topica viene aggiunto a inizio riga il timestamp corrente, il producer invia i dati alla topica alla massima velocità possibile.

3.2 2 tipi di latenza

Quando viene fatta una reduce andiamo ad unificare più righe del dataset che sono state inserite nella topica a tempi di processamento diversi, durante queste operazioni viene tenuta traccia del tempo di processamento più vecchio e più nuovo, quindi ogni output prodotto da flink e da kafka conterrà 3 tempi: event time,processing time riga più vecchia,processing time riga più nuova. Abbiamo la latenza del record più vecchio e la latenza del record più nuovo per ogni output da flink/kafka

3.3 Consumer

Il consumer non fa altro che leggere da una topica kafka (output di flink o kafka streams) registra il timestamp corrente ad ogni record prelevato da quest'ultima e calcola i due tempi di latenza in secondi andando a fare la differenza tra quello corrente e quelli presenti nell'output,

3.4 throughput

Per misurare il throughput ho utilizzato questo comando di kafka streams

```
$ kafka/bin/kafka-consumer-perf-test.sh
--topic output-stream
 --broker-list "localhost:9092,localhost:9093,localhost:9094"
 --messages 900 --threads 1
```

che tra le misure che ritorna vi è anche la latenza in termini di messaggi al secondo

throughput nMsg/sec	1 day	7 day	30 day
Query1 flink	7.3576	1.4208	0.3401
Query1 kafka	0.8268	1.1856	3.0889
Query2 flink	6.3028	1.2333	

3.5 Latenza

In queste tabelle mostriamo la latenza media,minima,massima su ogni cella sono presenti due valori il primo corrisponde alla latenza del record più recente (in termini di inserimento nella topica di input) e il secondo del record più vecchio (nella cartella /results/tempi sono presenti tutti i grafici delle latenze)

latenza media in sec	1 day new old	7 day new old	30 day new old
Query1 flink	0.496 0.640	0.726 1.418	3.679 5.730
Query1 kafka	176.675 176.845	183.018 183.921	130.153 133.146
Query2 flink	0.686 0.837	0.725 1.392	

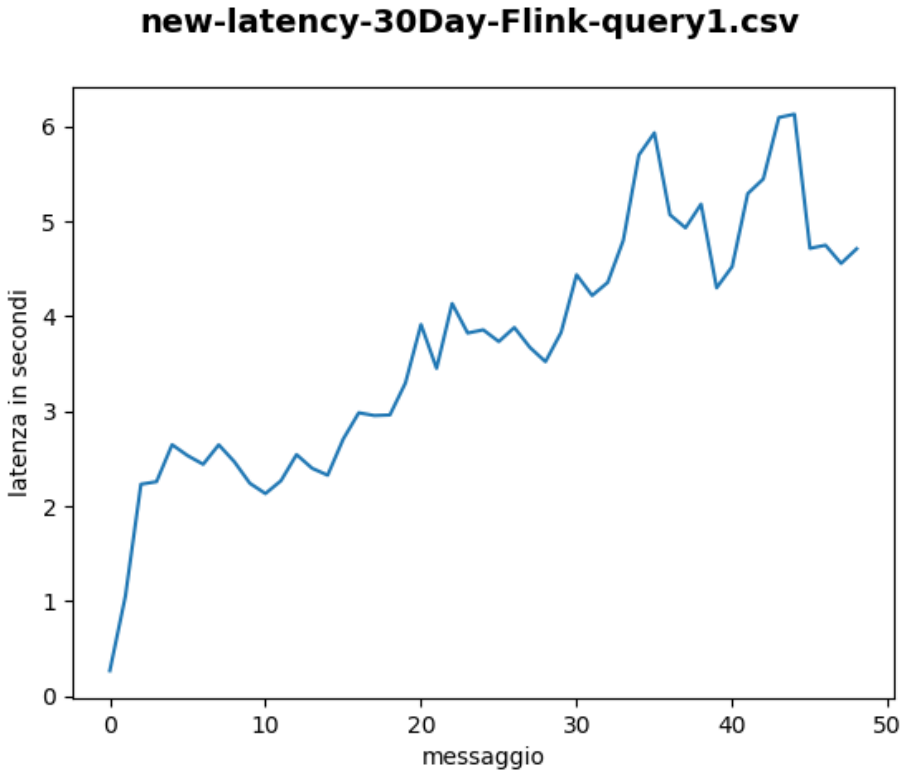
latenza minima in sec	1 day new old	7 day new old	30 day new old
Query1 flink	0.113 0.158	0.185 0.409	0.267 0.366
Query1 kafka	12.385 12.401	16.743 17.426	10.603 11.294
Query2 flink	0.347 0.430	0.310 0.340	

latenza massima in sec	1 day new old	7 day new old	30 day new old
Query1 flink	1.421 3.021	3.027 4.016	6.126 10.338
Query1 kafka	232.500 232.605	260.347 259.638	205.181 209.985
Query2 flink	2.515 3.653	2.678 3.374	

4 CONCLUSIONI

4.1 Apache Flink

Apache Flink si è dimostrato un framework semplice semplice da svilupparci e guardando i tempi anche efficente riuscendo a processare lo streamming del dataset aggiungendo poca latenza e con un alto throughput. Flink inizia a mostrare rallentamenti significativi con la finestra da 30 giorni avendo una latenza media del record più nuovo pari a 3.679 secondi e vediamo dal grafico che l'andamento della latenza è crescente



4.2 Kafka Streams

ho trovato Kafka Stream a livello di sviluppo più difficoltoso inoltre dai test effettuati non è riuscito a "rimanere dietro" allo stremming dei dati tantè che mediamente aveva una latenza superiore ai due minuti. Inoltre ho notato che kafka e flink hanno una gestione diversa delle

finestre, flink non processa la finestra fino a quando non viene superato l'event time di fine finestra,mentre kafka appena arriva un nuovo record processa tutta finestra e se arriva un novo record che rientra ancora nella stessa finestra la riprocessa, questo potrebbe essere un motivo per la differenza delle performance dei due framework. Una cosa interessante è osservare il grafico della latenza di kafka stream notiamo che la latenza arriva fino a un picco di oltre 200 secondi per poi iniziare a diminuire questo pattern è stato osservato in tutti i test fatti con kafka streams