Alma Mater Studiorum
Università di Bologna
Sede di Cesena

Assignment Big Data

Giovanni Mormone Fabián Andrés Aspée Encina

Professore : **Enrico Gallinuco**
Corso :**Big Data 19/20**

Agosto 2020

Indice

Lista di Figure

Lista di Tabelle

1 Introduzione

Il trattamento di grandi dataset richiede molte risorse per poter essere effettuato. Big Data ci permette di poter realizzare il trattamento di grandi dataset in una minor quantità di tempo, riducendo in questo modo le quantità di risorse impiegate, ed evitando di sprecare tempo.

Nel nostro caso, abbiamo scelto usare un dataset riguardante birre, birrerie e recensioni delle birre divisi in 3 file, con un peso totale 2 GB circa. L'idea principale è quella di sviluppare due job utilizzando due diverse tecnologie, in particolare, facendo uso di Hadoop (MapReduce) e Spark.

Map-Reduce è una componente di hadoop che ci permette poter processare grandi quantità di dati tramite un *map* e un *reduce*,in più permette di poter concatenare dei job, rendendo possibile utilizzare l'output di una reduce come input di un nuovo job e così via fino alla risoluzione del problema prefissato. Il principale *problema* che si può avere utilizzando MapReduce concatenando molti job, è che se abbiamo troppi dati, tende a essere più lento visto che scrive tutto su hard-disk.

Spark, che si appoggia su hadoop, permette di realizzare le sue stesse operazioni ma in modo più ordinato e ottimizzato, fornendo inoltre la possibilità di utilizzare un paradigma funzionale, laddove sia programmato in scala. Spark, a differenza di Hadoop MapReduce, invece di scrivere sull'hard-disk di default tiene tutto in memoria(se possibile),rendendolo più performante di Hadoop MapReduce.

Nelle seguenti sezioni verrà spiegato il modo in cui sono stati sviluppati i job e le difficoltà avute durante lo sviluppo di questi.

Il nostro lavoro viene organizzato nella seguente maniera, nella sezione 2 verranno spiegati i due job sviluppati in MapReduce e come è stato organizzato il lavoro, nella sezione 3 verranno spiegati i due job sviluppati usando Spark e come è stato organizzato il lavoro.

2 Descrizione Dataset e Jobs

Il dataset utilizzato per il progetto può essere trovato a questo link. Questo si compone di 3 file così suddivisi:

- beers.csv, in cui sono raccolte birre, di cui sono presenti tra i vari campi id, nome e idBirreria associata.
- breweries.csv, in cui sono raccolte le birrerie, di cui sono presenti tra i vari campi id e nome.
- reviews.csv, in cui sono raccolte le recensioni riguardo le birre, di cui sono presenti tra i vari campi l'id della birra recensita, vari voti e un overall che rappresenta la media tra i voti delle varie altre categorie.

I due job sviluppati sono così definiti:

- Job1: Top N birrerie (N passabile come parametro, default 20) con almeno N birre diverse (N secondo parametro, default 5) con le medie di voti piu alta.(N recensioni minime per ogni birra, terzo parametro, 50 default).
- Job2: Prima vengono classificate le birre in base alla media dei voti, poi vengono divise in classi di voto: media voto ≤ 2 bassa qualità; $2 <$ media voto ≤ 4 media qualità; media voto ¿4 alta qualità. Per ogni birreria viene calcolata la quantità di birre in ogni classe; alla fine, le birrerie vengono ordinate in base ad uno score, calcolato in base al numero di birre presente in ogni categoria, normalizzato rispetto al massimo numero di ogni classe di birre assegnate alle birrerie.

3 Hadoop

Per poter realizzare i task descritti nella sezione 2, abbiamo deciso di utilizzare dei ControlledJob, che ci permettono di aggiungere dipendenze tra i job che compongono il task. Questo è necessario perché per poter avviare alcuni job è necessario attendere che uno o più job siano finiti, per poter utilizzare i loro risultati come input. Oltre all'utilizzo di ControlledJob, abbiamo creato varie classi che implementano l'interfaccia Writable di Hadoop.io; questa interfaccia è necessaria per poter salvare sia il risultato di un Map che quello di un Reduce in una classe creata ad hoc per poter gestire la parte di dati a noi necessaria. Alcuni esempi di queste classi Writable sono la classe Beer, che al suo interno ha l'id, il nome e l'id della birreria delle birre; Brewery, che ha id e nome di una birreria. La classe BeerOrBrewery è invece rappresentativa di un tipo di classe che abbiamo utilizzato per poter fondere in un Reducer i risultati di Map effettuati su input diversi: viene utilizzata ad esempio per unificare le birre e le birrerie in un unica classe; il suo funzionamento si basa sulla presenza al suo interno di un'istanza della classe Beer, una della classe Brewery e di un campo booleano che distingue il tipo di classe che è presente dopo un Map. In questo modo durante la fase di Reduce, che ha come chiave l'id della birreria e come valore i vari BeerOrBrewery associati a essa, è possibile distinguere tra una birreria e le sue birre ed unificare il risultato. Tutte le strutture ricorrenti sono state inserite in

un package commonjob, dove oltre alle classi in comune sono presenti alcuni job che vengono eseguiti da entrambi i task(ad esempio la fusione tra birre e birrerie) e alcune configurazioni comuni ai due task, ad esempio il percorso del dataset, i percorsi di output e la gestione delle configurazioni e della creazione dei job.

3.1 Job1

Per lo sviluppo del primo task,che è stato spiegato nella sezione 2, abbiamo proceduto per step successivi: il primo job che viene eseguito si occupa di calcolare la media delle recensioni per ogni birra presente nel dataset reviews.csv: nella fase di map viene utilizzato l'id della birra come chiave mentre come valore è utilizzato l'overall della recensione. Nel Reducer invece viene semplicemente calcolata la media dei voti di ogni birra. Il secondo job si occupa di unificare le birre e le birrerie: la struttura del è spiegata nella sezione 3; da notare il fatto che per poter utilizzare due file diversi nella fase di Map è necessario utilizzare la classe MultipleInputs a cui associare i Mapper relativi ai file da processare; per poter salvare il risultato del Reduce su disco invece viene utilizzato un SequenceFileOutputFormat come formato di output, in quanto deve essere salvata una classe creata da noi che implementi l'interfaccia Writable. Dopo i primi due job viene eseguito il job che unifica i risultati dei primi due job: per poter essere eseguito questo job dipende dai primi due, quindi non può partire prima che essi terminino. Il job di unione utilizza anch'esso dei MultipleInputs, input che sono il risultato dei due job precedenti, e come output della fase di Map vengono restituite istanze della classe BreweriesAndAvg (che si basa sul funzionamento spiegato per la classe BeerOrBrewery nella sezione 3;) che hanno l'id delle birre come chiave. Il reducer invece si occupa semplicemente di unificare i risultati dei Map, avendo però l'accortezza di utilizzare come chiave del risultato l'id della birreria e come valore la media delle recensioni delle birre, che ci servono nell'ultimo job. L'ultimo job, che dipende sul precedente per poter essere eseguito, si occupa di leggere il risultato del job di unione e calcolare la media dei voti di ogni birreria, salvando il risultato finale in maniera ordinata. Per quanto riguarda la fase Map, esso legge semplicemente il risultato del precendte job e utilizza le chiavi delle birrerie come chiavi per mandare i dati al Reducer. Il reducer invece è la parte più elaborata di questo job: esso ha al suo interno due mappe, che hanno come chiave entrambe l'id della birreria, mentre come valore una ha il nome della birreria e l'altra la media dei voti delle recensioni delle birre. Durante la prima fase, nel metodo reduce, il Reducer si occupa di popolare queste due mappe; per poter effettivamente calcolare il risultato finale del task è invece necessario utilizzare il metodo cleanup, che viene richiamato da Hadoop dopo il reduce: nel cleanup ordiniamo la mappa delle medie, unifichiamo il risultato con la mappa dei nomi della birreria e salviamo il risultato di questa unione nel percorso finale del Task. Da notare il fatto che, per poter ordinare in modo corretto le birrerie è necessario che il reducer sia unico, in quanto le strutture utilizzate per salvare i risultati del metodo reduce devono essere tutti nello stesso posto.

3.2 Job2

Per lo sviluppo del secondo task,che è stato spiegato nella sezione 2, abbiamo proceduto anche per lui per step successivi: il primo job, come nel precedente task, è quello che si occupa di lavorare sulle recensioni e di calcolare la media; a differenza del Job1 però non sono le medie a venire salvate ma la classe di qualità della birra. Il secondo job è identico a quello del primo task, in quanto deve solo fondere i dataset di birre e birrerie. Il terzo job dipende dai due job precedenti: utilizza un MultipleInputs come input del job (gli input sono i risultati dei job precedenti). La fase di map associa alle chiavi salvate dai precedenti job, che sono gli id delle birre, un'istanza della classe BreweriesAndClasses (che si basa sul funzionamento spiegato per la classe BeerOrBrewery nella sezione 3;) così da associare unificare le birre/birrerie con la relativa classe di qualità ottenuta dalle recensioni. Il reducer si occupa semplicemente di unificare i dati del map e salvarli, utilizzando l'id della birreria come chiave. L'ultimo job si occupa di classificare le birrerie in base ad uno score, ottenuto considerando il numero di birre per ogni classe in ogni birreria. Nella fase di Map di questo job, i risultati del job precedente vengono raggruppati in base all'id della birreria ed alla classe associata ad ogni risultato: per poter salvare una chiave composta abbiamo creato una classe Pair, che implementa l'interfaccia WritableComparable¡Pair¿, necessaria a poter utilizzare una classe come chiave di un Map; il valore associato alle chiavi è invece il nome della birreria. La fase di reduce è simile alla fase di reduce dell'ultimo job del primo task: in questo caso salviamo in una mappa ,con chiave il nome della birreria, un'istanza di una classe ResultToPrint definita nel Reducer. Questa classe si occupa di mantenere il numero di birre delle varie qualità associate ad ogni birreria. Nella fase di cleanup viene effettuato un ordinamento in base allo score e, in caso di pareggio in base al numero di birre di qualità alta prima, media poi e bassa alla fine. Per il calcolo dello score abbiamo normalizzato i valori in base al massimo numero di birre per ogni classe: è quindi necessario mantenere il conto del massimo di birre per ogni classe durante il reduce per poter in seguito normalizzare il risultato. Per questo motivo anche l'ultimo job di questo task ha necessità di utilizzare un singolo reducer.

4 Spark

Come descritto nella sezione 3, per poter realizzare i task già nominati, abbiamo creato delle classi in comune, per esempio, sia il job1 che il job2 fanno uso delle classi Beers, Breweries, Reviews, allora queste tre classi vengono messe dentro un package common, che permette che entrambi job possano fare uso di queste senza doverli duplicare, in più è stata definita una classe common, che tiene i metodi comuni tra i job, come per esempio i percorsi di input, la verifica dei directory e il metodo per calcolare la media delle recensioni.

La SparkSession non è potuta essere messa in una sola classe perché Spark ha bisogno che la sua inizializzazione sia fatta nel main altrimenti può produrre delle eccezioni in tempo di esecuzione.

4.1 Job1

Per lo sviluppo del primo task, che è stato spiegato nella sezione 2, abbiamo proceduto per step successivi: inizialmente si crea una SparkSession e si verifica se ci sono dei valori passati come argomento, se ci sono vengono settati per passarli dopo al metodo che esegue le operazioni per il task da eseguire, nel caso non ci siano, vengono settati dei valori per default per ognuno di queste variabili. Prima di dar inizio al caricamento dei file, verifichiamo che non esistano le directory di output, visto che Spark le crea automaticamente, dopo questo controllo si procede a leggere i dataset che si trovano nel file system di Hadoop.

Procedendo, togliamo la prima riga di ogni file che fa riferimento all' header di questo e dopo li mappiamo trasformando questi in delle case class che ci permetteranno di poter manipolare i dati in un modo molto più comodo.

Sia Beers che Brewery hanno come chiave idBrewery, questo per poter far la unione tra di loro, mentre Reviews ha come id l'id della birra.

Dopo di aver fatto le operazioni nominate in precedenza, passiamo a filtrare le breweries, eliminando tutte le brewery che abbiano meno di 5 birre (parametro che può essere modificato tramite command line). Dopo questa operazione si procede ad unificare Beers con Brewery, per avere un RDD unico con tutte le birre e birrifici, una volta completato questo passaggio procediamo al calcolo della media per ogni birra, le cui informazioni sono presenti in Reviews, considerando come minimo 50 recensioni per birra se non specificato diversamente da command line. Procedendo con le operazioni, si calcola la media di tutte le birre in base alla loro recensione e col risultato si va alla ricerca delle top N birrerie, dove il RDD ottenuto dopo il join fatto tra Beer e Brewery viene joinato con il risultato appena ottenuto, in questo modo avremo birrificio, birra e la sua corrispettiva media che ci permette di poter fare nuovamente la media tra tutte le birre appartenenti ad un birrificio e dopo poterli ordinare in base al risultato e filtrare gli N birrifici (dove N se non è specificato tramite command line, prende come valore di default 20), per dopo essere salvato come un file di testo.

4.2 Job2

Per lo sviluppo del secondo task, che è stato spiegato nella sezione 2, abbiamo proceduto per step successivi: inizialmente si crea una SparkSession e si procede a verificare se ci sono dei parametri passati come argomento, se ci sono vengono settati per poterli passare al metodo che esegue le operazioni per il task da eseguire, nel caso in cui l'utente non specifichi nessun parametro, allora questi vengono settati con valori per default.

Come descritto nel job precedente 4.1, prima verifichiamo se esiste la directory di output, nel caso già esista questa viene eliminata, leggiamo i file da utilizzare togliendo la prima riga che rappresenta l'header,ognuno viene mappato in una case class che ci permetterà realizzare delle operazioni in modo più comodo, dopo

aver realizzato queste operazioni vengono filtrate tutte le birre che hanno più delle recensioni specificate dall'utente, in caso contrario si utilizza per default 50, dopo questo facendo uso del *aggregateByKey* facciamo il conteggio di tutte le birre e il loro score per dopo avergli fatto un mapValues per fare la divisioni di questi valori.

Una volta che l'operazione precedente è finita, vengono create delle classi per ogni risultato dove queste rappresenteranno la qualità delle birre. Dopo facciamo l'unione tra le birre e le brewery, in questo modo tutti birrifici senza birre vengono tolti.

In più si contano le quantità di birre per birrifici, tenendo conto dello score dato a ogni birra, cosi ogni birrificio terrà 3 classi che permetterà di poter assegnare delle birre in base al loro punteggio. In questo modo potremo calcolare il loro final score che viene calcolato considerando il valore massimo tra ogni classe e dividendo ogni valore dentro la classe per il massimo, da normalizzare tutti i valori ed ottenere dei risultati che fluttuano di meno, cioè vengono portati nel range tra [0-1], dopo questo lo scorre verrà dato da:

$$(lowNormalized * LOW\_SCORE) + (middleNormalized * MEDIUM\_SCORE)$$
$$+(highNormalize * HIGH\_SCORE)$$

4.3 Diagramma Attività Job2

La immagine 4.1 mostra a grandi linee gli steps che vengono seguiti per poter realizzare il job2 e le relazioni tra di loro:

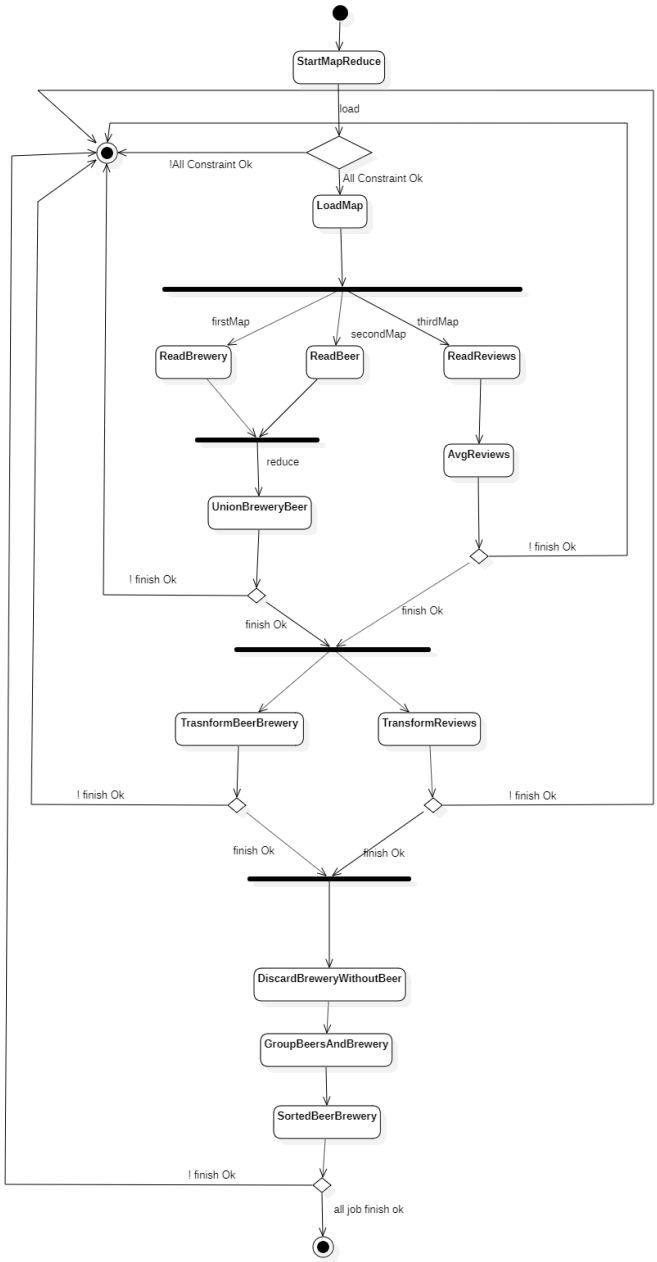

Figura 4.1 Diagramma Attivita Job2