Report on Big Data project: GoodBooks-10K-Jobs

Lorenzo Valgimigli - Mat.
Riccardo Soro - Mar.

July 22, 2019

Contents

1 Introduzione **3**
- 1.1 Descrizione Dataset . . . 3
  - 1.1.1 Descrizione File . . . 3

2 Preparazione dei dati **4**

3 Jobs **4**
- 3.1 Job1: Best 500 Authors . . . 6
  - 3.1.1 Pianificazione . . . 6
  - 3.1.2 Hadoop MapReduce . . . 6
  - 3.1.3 Hadoop Jobs History . . . 8
  - 3.1.4 SetUp SparkSql . . . 8
  - 3.1.5 Implementazione sparksql . . . 8
  - 3.1.6 Conclusioni . . . 9
- 3.2 Job2: Correlazione tra rating e bookmarks . . . 10
  - 3.2.1 Pianificazione . . . 10
  - 3.2.2 Hadoop MapReduce . . . 10
  - 3.2.3 Implementazione . . . 11
  - 3.2.4 Hadoop Jobs History . . . 12
  - 3.2.5 SetUp SparkSql . . . 12
  - 3.2.6 Implementazione sparksql . . . 12
  - 3.2.7 Conclusioni . . . 13

4 Miscellaneous **14**
- 4.1 Problematiche ottimizzazione . . . 14
- 4.2 Problematica Job1 . . . 14

1 Introduzione

1.1 Descrizione Dataset

Il dataset scelto viene fornito dal negozio online di libri Good Reads ed è reperibile sul repository github GoodBooks-10k. All'interno possiamo trovare una serie di file `.csv` che contengono 10000 libri con vari attributi, oltre 6 milioni di ratings e qualche centinaia di bookmarks. In totale il data set raggiunge circa 95 MB. Colleziona 10000 libri, 53424 utenti e oltre 6 milioni di rating.

1.1.1 Descrizione File

Il dataset è composto dai seguenti file:

- `books.csv`: [lvalgimigli@isi-vclust9.csr.unibo.it:./exam/dataset/books.csv] Contiene metadati dei vari libri come:
  - `book_id`: ID del libro nel dataset
  - `goodreads_book_id`: ID del libro nel Data Store del negozio
  - `best_book_id`: ID della versione del libro che il negozio ha in vendita
  - `work_id`: ID per gastire dove il libro si trova nel magazzino
  - `books_count`: Numero di libri nel magazzino
  - `isbn`
  - `isbn13`
  - `authors`: Lista di autori separati da una virgola
- `ratings.csv`: [lvalgimigli@isi-vclust9.csr.unibo.it:./exam/dataset/ratings.csv] Contiene i rating fatti da tutti gli utenti. Le colonne di questa tabella sono:
  - `user_id`: ID dell'user che ha fatto la votazione
  - `book_id`: ID del libro nel dataset
  - `rating`: Valore da 1 a 5 che rappresenta il rating dell'utente per quel libro
- `to_read.csv`: [lvalgimigli@isi-vclust9.csr.unibo.it:./exam/dataset/to_read.csv] contiene i desideri degli utenti di leggere un libro. Questo desiderio è espresso con un *bookmark* la tabella è così composta:

  - `user_id`
  - `book_id`

- `book_tags.csv`: [lvalgimigli@isi-vclust9.csr.unibo.it:./exam/dataset/book_tags.csv] Questa tabella contiene per ogni libro i tags assegnatoli dagli utenti
  - `goodreads_book_id`
  - `tag_id`: ID del tag
  - `count`: Numero di utenti che hanno assegnato quel tag a quel libro

- `tags.csv`: [lvalgimigli@isi-vclust9.csr.unibo.it:./exam/dataset/tags.csv] Tabella che salva l'associazione tra tag ID e il nome del tag
  - `tag_id`
  - `tag_name`

2 Preparazione dei dati

Il referente del gruppo è lo studente Lorenzo Valgimigli. Tutti i file saranno presenti nella sua home.

Il dataset per l'esame si trova sià nel nodo `vclust9` nella home di `lvalgimigli` all'indirizzo `isi-vclust9.csr.unibo.it//home/lvalgimigli/exam/dataset/` I dati venivano indicati dal sito come completi e puliti per tanto non è stata necessarie una ulteriore fase di pulizia dati. E' stato solo sufficente dedicare del tempo ad capire il dominio e il significato degli attributi delle varie tabelle.

3 Jobs

L'obbiettivo del progetto è quello di realizzare 2 jobs utilizzando sia `Hadoop MapReduce` che `Spark`. In quest'ultimo caso è stato scelto di utilizzare `Spark-Sql`. Di seguito vengono esposti i Jobs:

1. **Job1**: Trovare i 500 migliori autori nel dataset. Ovvero, trovare i 500 autori la cui media di punteggio ottenuto nei loro libri è più alto. Per tanto sarà necessario calcolare per ogni libro il rating medio. Poi sarà necessario associare i libri agli autori e calcolare la media del rating dei vari libri.

2. **Job2**:Trovare una correlazione tra numero di *bookmarks* e *rating medio* dei vari libri. Sarà quindi necessario calcolare per ogni libro i bookmarks e il rating medio.

E' bene sottolineare che già dalla fase di analisi dei requisiti si evince un problema che dovrà essere gestito: il campo `author` in `books.csv` è composto da una lista di autori per tanto sarà necessario dividerli. Per sviluppare il progetto si è scelto di creare un progetto `gradle` usando `Java` per la parte di `Hadoop MapReduce` mentre si è scelto di utilizzare `Scala` per la parte di `SparkSQL`

3.1 Job1: Best 500 Authors

In questo primo esercizio lo scopo consiste nel trovare i 500 autori che hanno le medie delle valutzioni più alte. I punteggi vanno da 1 a 5 e sono assegnati dagli utenti ai libri, mentre per ogni libro è presente la lista di autori che partecipato alla sua scrittura.

3.1.1 Pianificazione

Le tabelle che entrano in gioco per questo job sono le seguenti:

- `ratings.csv`
- `books.csv`

La tabella ratings è composta dall'id dell'utente, l'id del libro e dalla valutazione, quest'ultima varia da 1 a 5. La chiave primaria in questa tabella è la combinazione id_libro e id_utente e ci serve per calcolare la media delle valutazioni per ogni libro e, successivamente, calcolare la media del punteggio medio di ogni libro scritto da ogni attore. Per fare questa operazione si è resa necessaria la tabella books.csv, nella quale sono presenti gli autori che hanno partecipato alla scrittura dei libri. In questa seconda tabella sono presenti moltissimi attributi, ma gli unici che servono sono la coppia id_book (chiave primaria della tabella) e la lista di autori (separati da ;).

Per risolvere il problema occorre come prima cosa effettuare un join tra le due tabelle sull'attributo id_book, comune ad entrambe. Occorre inoltre calcolare la media delle recenzioni di ogni libro, per poi dare un punteggio ad ogni autore pari alla media delle valutazioni di tutti i libri sui quali ha lavorato. Una volta ottenuta una tabella con attributi autore - rating_medio manca solo la fase di sorting per ordinare i risultati.

3.1.2 Hadoop MapReduce

Per svolgere l'esercizio è stato necessario implementare più job, è stato possibile ridurne il numero fino a tre job distinti tra loro. Nello specifico il primo di questi si occupa di fare il join tra le due tabelle, il secondo svolge un operazione di raggruppamento e l'ultimo ordina correttamento e globalmente gli elementi.

- **Fase di Join** : Per questa fase è stato definito un job che esegua il join delle due tabelli sulla chiave id_book. Questo job è composto da due Mapper e un Reducer, i mapper (uno a tabella) si occupa di

leggere la tabella, escludere gli attributi senza utilità e di creare come output una lista di coppie id_book - attributo. Dalla tabella books.csv viene estrapolato l'attributo authors e dalla seconda il rating; il reducer si occupa invece di raccogliere questi output, calcolare la media delle votazioni di ogni libro per poi produrre un output composto dalla lista di coppie contenenti singolo_autore - rating_medio_per_singolo_libro. L'output perciò potrà contenere più volte lo stesso autore, nello specifico ogni autore sarà presente tante volte quante sono le volte in cui appare come collaboratore nei libri.

- **Fase di Filtering** : Nella fase precedente si sono ottenuti in output risultati incompleti, con autori che si ripetono e con valori di valutazione diversi. La fase di Filtering ha come obiettivo quello di ottenere un output completo di tutte le informazioni richieste dall'esercizio senza avere però informazioni ridondanti o non necessarie. L'implementazione di questa fase è composta da da un mapper e un reducer, il primo elabora l'output della fase precedente creando in uscita una lista contenente autore - rating_medio_per_singolo_libro, la quale sarà ulteriormente elaborata dal reducer al fine di ottenere per ogni autore la media delle medie dei libri da lui scritti. Grazie a questa fase si ottiene in output tutti i valori necessari alla risoluzione dell'esercizio, ma essendo l'output suddiviso in più file (uno a reducer) non possiamo in questa fase ottenere un output ordinato globalmente.

- **Fase di Sorting** : Nella fase precedente si sono ottenuti come output risultati disordinati globalmente. Per risolvere un ploblema di sorting globale è stato utilizzato inizialmente un partitioner personalizzato, basato su soglie del rating_medio che garantissero una distribuzione omogenea dei dati sul reducer garantendo l'ordinamento grazie appunto alle soglie prestabilite. Questa implementazione però non poteva funzionare con un numero variabile di reducer, il numero di possibili soglie utilizzabili erano limitate; è stato perciò deciso di utilizzare il TotaOrderPartitioner. Questo metodo calcola in automatico le soglie partendo da un numero specificabile di record, permettendo al job di poter essere eseguito su un numero qualsiasi di reducer per non limitare la scalabilità della soluzione. In uscita da questo job si ottiene quindi la lista degli autori e del loro rating_medio ordinati dal rating più alto a quello più basso.

Il comando per lanciare l'eseguibile è il seguente

```
hadoop jar GoodBooks-10k-Jobs-2.1.2-mr1.jar
```

Va lanciato dalla directory: `/home/lvalgimigli/exam/mapreduce`

3.1.3 Hadoop Jobs History

Di seguito i link alle history dei job su `YARN`

- `http://isi-vclust0.csr.unibo.it:19888/jobhistory/job/job_1560510165054_1738`
- `http://isi-vclust0.csr.unibo.it:19888/jobhistory/job/job_1560510165054_1739`
- `http://isi-vclust0.csr.unibo.it:19888/jobhistory/job/job_1560510165054_1740`

3.1.4 SetUp SparkSql

Per ottimizzare le performance di SparkSQL sul nodo del cluster sono stati settati i seguenti parametri:

- `spark.default.parallelism` = 8
- `spark.sql.shuffle.partitions` = 8
- `spark.executor.memory` = 11g

3.1.5 Implementazione sparksql

Il Job è stato realizzando utilizzando a pieno SparkSQL affidandoci alle ottimizzazioni interne del framework. La risoluzione è contenuta nel file `it.unibo.bd1819.job2.MainJob2`. Il problema stato affrontato tramite i seguenti step:

- `Creazione DataFrame`. Come primo passo abbiamo sfruttato una feature di `spark2` per leggere i dati dal file .csv inferendo direttamente lo schema della tabella. Una volta letto il file viene automaticamente creato un oggetto DataFrame pronto all'uso.
- `Split degli autori`: In questa fase volevamo che una tupla con il campo autori contenente una lista di autori diventasse un insieme di tuple con il campo autore contenente un singolo autore. Per fare questo, dopo aver caricato il file .csv nel DataFrame abbmiamo creato una funzione di `split` (SplitAuthors nel Package Object **bd1819**). Poi nel file MainJob2 abbiamo utilizzato una funzione di flatMap in cui gli abbiamo specificato la funzione di split precedentemente creata.

- `Rating medio`. Per calcolare il rating medio non abbiamo bisogno di fare Join con altre tabelle ma ci basta utilizzare ratings. E' stata quindi utilizzata la seguente query per calcolare il rating:

```
SELECT book_id, AVG(rating) as avgRating FROM ratings
GROUP BY book_id
```

- `Join`: E' quindi necessario fare il Join tra le tabelle books.csv e quella calcolata nel passo precedente. Entrambe sono relativamente piccole per cui si è scelto di utlizzare esplicitamente il metodo Broadcast Join usando come tabella Broadcast `books.csv` selezionando solo i campi necessari.

- `Media ratings dei vari autori`: Una volta effettuato il Join sarà necessario calcolare la media dei rating di ogni libro per ogni autore. Per fare ciò è stata utilizzata la seguente Query:

```
SELECT  authors, avg(average_rating) as avg FROM finalview
GROUP BY authors ORDER BY avg desc
```

- `Visualizzazione dei risultati`: Una volta calcolati tutti i valori necessari la tabella risulta essere già ordinata quindi per ottenere il risultato finale basterà stampare le prime 500 righe.

Per lanciare il job bisognerà andare nella cartella `/home/lvalgimigli/exam/sparksql` e lanciare il jar con il seguente comando:

```
spark2-submit GoodBooks-10k-Jobs-2.1.2-spark.jar JOB1
```

3.1.6 Conclusioni

Realizzare questo job in sparksql è risultato essere più semplice è più performante rispetto Hadoop. Combinando la potenza semantica di scala e la semplicità di sparksql si è riuscito con poce linee di codice a risolvere problemi complessi come lo splitting degli autori.

3.2 Job2: Correlazione tra rating e bookmarks

L'obbiettivo di questo Job è quello di vedere se esiste una correlazione tra rating medio e bookmarks Quindi il cuore del Job è il calcolo del rating e dei bookmarks e fare un confronto incrociato.

3.2.1 Pianificazione

Le tabelle che entrano in gioco per questo job sono le seguenti:

- `ratings.csv`
- `to_read.csv`

La prima verrà utilizzata per calcolare il rating medio mentre la seconda verrà utilizzata per calcolare i Bookmarks. Dopo di che i risultati ottenuti saranno uniti in una unica tabella e da li verranno calcolati i risultati.

Ogni libro può quindi appartenere a uno dei 4 gruppi:

- *Molti Bookmarks e alto rating*
- *Molti Bookmarks e basso rating*
- *Pochi Bookmarks e basso rating*
- *Pochi Bookmarks e alto rating*

3.2.2 Hadoop MapReduce

Per risolvere l'esercizio sono state calcolate empiricamente le due soglie da utilizzare per classificare i libri. Una volta ottenute queste soglie è stato possibile pianificare il numero di job necessari e il loro funzionamento. Il problema si può dividere in quattro fasi:

- **Calcolare il rating** In questa fase bisogna leggere la tabella rating, eliminare gli attributi non necessari e calcolare la media delle valutazioni per ogni libro.
- **Calcolare i bookmarks** In questa fase bisogna leggere la tabella to_read, eliminare gli attributi non necessari e calcolare la somma del numero degli utenti che desiderano leggere il libro,tutto questo per ogni libro.

- **Classificare per rating** In questa fase si classificano i libri comparando il rating calcolato precedentemente con la soglia ricavata empiricamente.
- **Classificare per bookmarks**In questa fase si classificano i libri comparando il numero degli utenti che li desiderano leggere con la soglia ricavata empiricamente.
- **Conteggio delle classi**In questa fase si conta il numero di record delle varie classi al fine di determinarne le più frequenti.

In fase di implementazione è stato possibile raggruppare questi step al fine di ottenere un numero minore di job, di conseguenza raggiungere livelli di performance migliori. L'esercizio è stato svolto aggregando le prime quattro fasi nel job computeAVGRatingJob, mentre l'ultima è stata implmentata in counterJob.

- **computeAVGRatingJob** Questo job legge entrambe le tabelle e realizza le fasi uno e due, per poi fare i calcoli delle fasi tre e quattro. Il jo si conclude con la creazione dell'output, composta dalla semplice classe di appartenenza di ogni libro.
- **counterJob** Questo job si occupa della lettura del file generato nel job precedente e di contare i valori in esso contenuto al fine di creare come file di output un file contenente il totale di libri per ogni classe.

3.2.3 Implementazione

Per implementare il primo job si è reso necessario lo sviluppo di due mapper, uno per ogni tabella. Entrambi i mapper leggono dal file gli attributi rilevanti escludendo quelli non necessari al completamento dell'esercizio. Nello specifico il mapper relativo alla tabella rating.csv crea come output una lista di book_id - rating per ogni libro, mentre l'altro produce una lista di book_id per ogni volta che quel libro è presente nella lista dei libri da leggere di un utente. Il reducer che è incaricato della gestione dell'output di questi due mapper è CounterRatingReducer, il quale si occupa anche della classificazione dei libri creando un output contenente la classe di appartenenza di ogni libro.

Il secondo job si occupa semplicemente di contare il numero di libri che vengono classificati nelle quattro possibili classi. Il mapper utilizzato da questo job è CountMapper e come chiave dell'output utilizza la classe di appartenenza del libro. Il reducer utilizzato è CountReducer e scrive sul file

di output il numero di volte che un libro è presente in una classe. Per questo secondo job è possibile sfruttare al massimo quattro reducer, la causa è il numero di classi che possono avere i libri. Il mapper infatti può produrre al massimo quattro tipi diversi di chiave, di conseguenza qualsiasi funzione si utilizza per il partizionamento creerà al massimo quattro risultati diversi, limitando di fatto il numero di reducer che si possono sfruttare.

Il comando per lanciare l'eseguibile è il seguente

```
hadoop jar GoodBooks-10k-Jobs-2.1.2-mr1.jar
```

Va lanciato dalla directory: `/home/lvalgimigli/exam/mapreduce`

3.2.4 Hadoop Jobs History

Di seguito i link alle history dei job su `YARN`

- `http://isi-vclust0.csr.unibo.it:19888/jobhistory/job/job_1560510165054_1741`
- `http://isi-vclust0.csr.unibo.it:19888/jobhistory/job/job_1560510165054_1742`

3.2.5 SetUp SparkSql

Per ottimizzare le performance di SparkSQL sul nodo del cluster sono stati settati i seguenti parametri:

- `spark.default.parallelism` = 8
- `spark.sql.shuffle.partitions` = 8
- `spark.executor.memory` = 11g

3.2.6 Implementazione sparksql

Per questo problema si utilizzato a pieno SparkSQL affidandoci alle ottimizzazioni interne del framework. Il problema stato affrontato tramite i seguenti step:

- `Creazione DataFrame`. Come primo passo abbiamo sfruttato una feature di `spark2` per leggere i dati dal file .csv inferendo direttamente lo schema della tabella. Una volta letto il file viene automaticamente creato un oggetto DataFrame pronto all'uso.
- `Rating medio`. Per calcolare il rating medio non abbiamo bisogno di fare Join con altre tabelle ma ci basta utilizzare ratings. E' stata quindi utilizzata la seguente query per calcolare il rating:

```
SELECT book_id, AVG(rating) as avgRating FROM ratings
GROUP BY book_id
```

- `Bookmarks`. Anche per calcolare i bookmarks bancia lanciare una query su una sola tabella:

  ```
  SELECT book_id, count(user_id) as marks FROM bookmarks
  GROUP BY book_id
  ```

- `Join tra i risultati`. Ora bisogna unire i due risultati in una unica tabella. Per il join essendo le tabelle abbastanza piccole è stato usato un `broadcast join` sulla tabella con i rating.
- `Risultati`. Ora abbiamo una tabella con tutte le informazioni che ci servono. Ci basterà lanciare qualche query che filtra i risultati per rating e bookmarks e vedere quanti elementi sono in ogni gruppo.

Per lanciare il job bisognerà andare nella cartella `/home/lvalgimigli/exam/sparksql` e lanciare il jar con il seguente comando:

```
spark2-submit GoodBooks-10k-Jobs-2.1.2-spark.jar JOB2
```

3.2.7 Conclusioni

Realizzare questo task in SparkSQL è stato estremamente semplice sià per la semplicità in se del task sia per l'interfaccia intuitiva di SparkSQL. Il risultato finale viene calcolato in tempi abbastanza buoni grazie alle ottimizzazioni interne del framework e al setup generale.

4 Miscellaneous

4.1 Problematiche ottimizzazione

Il problema che abbiamo riscontrato per quanto concerne l'ottimizzazione è la dimensione del DataSet. Il dataset è di dimensioni relativamente ridotte: sotto i 100 MB. Per tanto l'ottimizzazione fatta dal framework SparkSQL di default è già molto buona. Sono stati provati vari settaggi per provare ad apportare delle migliorie ma la piccola dimensione non ha permesso l'apprezzamento di sostanziali differenze.

4.2 Problematica Job1

Per lo sviluppo del Job1 utilizzando il modello MapReduce non si è considerato opportuno allocare un intero Job per selezionare i primi 500 autori, infatti sarebbe stata un'operazione pesante per il modello MapReduce mentre un semplice script la potrebbe farlo con un costo computazionale decisamente più piccolo. Ordinarli secondo il rating medio è stato considerato sufficente.