Report del progetto di Big Data: analisi dei crimini nella città di Chicago

Alessia Ventani - Mat. 901809
Simone Venturi - Mat. 907547

13 luglio 2020

Indice

1 Introduzione

1.1 Descrizione del dataset

Per questo elaborato si è deciso di utilizzare gli open data che la città di Chicago ha reso accessibili on-line dal suo sito. Fra i vari dataset presenti, ci si è concentrati su quello riportante i crimini commessi nella città Chicago dal 2001 ad oggi, aggiornati a sette giorni precedenti la data del download. I dati sono estratti dal CLEAR, acronimo di *"Citizen Law Enforcement Analysis and Reporting"*, del dipartimento di polizia della città. Per motivi ovvi di privacy, i nomi propri sono omessi e gli indirizzi non riconducono ad una specifica posizione geografica ma ad un'area, più o meno grande in base al grado di granularità del campo scelto per l'ubicazione geografica.
I dati utilizzati sono scaricabili ai seguenti link:

- elenco dei crimini registrati: `https://data.cityofchicago.org/Public-Safety/Crimes-2001-to-present/ijzp-q8t2`
- elenco dei codici univoci di report dei crimini dello stato dell'Illinois: `https://data.cityofchicago.org/Public-Safety/Chicago-Police-Department-Illinois-Uniform-Crime-R/c7ck-438e`

Dai questi è possibile scaricare i dati premendo sul tasto Export e scegliendo il formato "CSV Excel for Europe".

1.1.1 Descrizione dei file

L'analisi si basa sull'utilizzo di un unico file csv di partenza con i dati dei crimini registrati a Chicago. Nella tabella sono presenti 22 colonne che riportano informazioni di varie categorie: i dettagli sulla tipologia di reato, la sua collocazione temporale e spaziale (comprese le coordinate), se è stato effettuato un arresto o meno e se il crimine è avvenuto in casa o meno. Di queste colonne sono stati considerati solo alcuni campi.
Per il primo job sono stati utilizzati:

- IUCR: codice univoco di identificazione di un crimine per lo stato dell'Illinois;
- Description: breve descrizione del crimine riportato;
- District: codice corrispondente al distretto di polizia in cui è avvenuto il crimine.

Per la seconda elaborazione ci si è concentrati su:

- IUCR: codice univoco di identificazione di un crimine per lo stato dell'Illinois;
- Description: breve descrizione del crimine riportato;

- Arrest: booleano che indica se per il crimine è stato effettuato un arresto o meno;
- Year: anno in cui è avvenuto il crimine.

Un esempio dei dati utilizzati è:
IUCR Description District Arrest Year
0460 SIMPLE 006 False 2020

2 Preparazione dei dati

I dati considerati non hanno avuto bisogno di operazioni di pre-elaborazione complesse data la loro struttura iniziale a tabella. Si noti però che all'interno del dataset, essendo compilato manualmente, potrebbero esserci degli errori accidentali o omissioni di dati ma questo non costituisce un grosso problema per le elaborazioni che si intendono eseguire.

Per poter effettuare delle valutazioni sulle performance nel primo job, vedi descrizione nella sezione successiva, sono state preparate due versioni del file inziale. La prima costituisce il file scaricato dal sito nella sua versione integrale (`user/aventani/progetto_ventani/2/Crimes_-_2001_to_present.csv`) mentre per la seconda, si è deciso di dividere quest'ultimo in due:

- una prima tabella comprendente tutti i campi ad eccezione di "Primary Type" e "Description" `user/aventani/progetto_ventani/2/Crimes-2001-to-present-without-description.csv`. Per ottenere questo file è stato fatto eseguire il seguente codice spark:

```
spark.read.format("csv")
  .option("sep", ";").option("header", "true")
  .option("mode", "DROPMALFORMED")
  .load(<path to input file>)
  .drop("Primary Type","Description").coalesce(1)
  .write.format("com.databricks.spark.csv")
  .option("sep", ";").option("header", "true")
  .save(<path to output file>)
```

- una seconda riportante i campi "IUCR","Primary Type" e "Description", `user/aventani/progetto_ventani/2/Chicago_Police_Department_Illinois_Uniform_Crime_Reporting_IUCR.csv`. In questo caso non è stato necessario elaborare il file di partenza, ma è stato semplicemente scaricato l'elenco dei codici univoci di report dei crimini dello stato dell'Illinois. Unico problema riscontrato per il confronto dei dati è stato l'assenza in alcuni codice IUCR dello zero iniziare. Si è resa necessaria, quindi per farli coincidere con quelli presenti nella prima tabella, l'aggiunta di questa cifra,

operazione effettuata manualmente dato il basso numero di record con questa caratteristica.

Come formato si è deciso di caricare il file csv e utilizzare come separatore il punto e virgola, *";"*, per poter distinguere il simbolo di separazione delle colonne da eventuali virgole presenti nelle descrizioni dei crimini.

3 I Job

Per avere un riscontro dei risultati e verificarli si è deciso di suddividere i due task fra i componenti del gruppo nella maniera seguente: il primo job è stato realizzato da Alessia Ventani in MapReduce mentre da Simone Venturi in spark, per il secondo invece sono state scambiate le tecnologie quindi Ventani lo ha realizzato in spark mentre Venturi in MapReduce.

3.1 Job #1: conteggio dei crimini per distretto

Lo scopo di questo job è quello di restituire la lista del crimini, raggruppati per distretto e per tipologia, con il relativo conteggio delle registrazioni effettuate. I record risultanti devono essere ordinati in ordine crescente per numero di distretto e in ordine decrescente per totale di crimini registrati. Dunque un esempio di del risultato che deve essere ottenuto è:

```
001     FROM BUILDING 30616
001     $500 AND UNDER 30127
...
002     $500 AND UNDER 29150
002     SIMPLE 28368
...
```

In questi, il primo codice rappresenta il codice univoco del distretto del polizia dove è stato registrato il crimine, colonna District, la seconda parte rappresenta una breve descrizione, colonna Description, e l'ultimo numero è il totale dei crimini avvenuti in quel distretto e con quella descrizione presenti nel database.

In questo task si è deciso di effettuare l'elaborazione in due modalità differenti. Nella prima si è deciso di mantene la tabella originale e raggruppare per IUCR e per Description, nella seconda si è divisa la tabella originale in due, come descritto nel paragrafo "Preparazione dei dati" e effettuare le operazioni sul IUCR per poi fare in un secondo momento il join con la tabella Description. L'obbiettivo di questa doppia modalità è quello di verificare se ci possa essere o meno un miglioramento delle performance con una versione rispetto all'altra nei due paradigmi utilizzati.

I file eseguibili per questo job, una volta compilato con gradlew, è nel jar BDE-mr-Ventani-Venturi.jar, per MapReduce, mentre in BDE-spark-Ventani-Venturi.jar per la versione in spark.

3.1.1 MapReduce

L'implementazione di questo job in MapReduce è stata eseguita da Alessia Ventani. Il codice prodotto nel paradigma MapReduce, presente nella classe CrimesCountDistrict, può essere eseguito, dopo essere stato compilato, con il seguente comando:

```
hadoop jar BDE-mr-Ventani-Venturi.jar CrimesCountDistrict
Crimes_-_2001_to_present.csv output-conteggio
output-count-no-join
```

Da questo è possibile visualizzare in un solo colpo i file che costituisco l'input e l'output di questo job. L'unico file di input è CrimesCountDistrictCrimes_-_2001_to_present.csv che, come detto in precedenza, rappresenta la tabella completa dei crimini dal 2001 ad oggi della città ci Chicago. I percorsi che costituiscono i parametri 1 e 2 del comando sono i due file di output: il primo contiene il risultato intermedio della conta del numero totale di crimini commessi in un distretto e di un determinato tipo, il secondo è il risultato finale del job e quindi il risultato è osservabile al percorso `user/aventani/progetto_ventani/output-count-no-join`. Per ottenere il risultato voluto, si è deciso di dividere la computazione in due fasi di MapReduce. Nella prima si effettua il conteggio del totale del numero di crimini di una determinata tipologia in un distretto, successivamente, nella seconda si ordina il risultato ottenuto in ordine crescente per il codice del distretto e in ordine decrescente per numero totale di crimini. Di seguito una breve descrizione le fasi principali:

- nel primo mapper il valore della chiave è costituito dalla tripla di valori District, IUCR e Description mentre il valore è un semplice uno;
- nel primo reducer, si sfrutta il meccanismo di partizionamento di MapReduce che fa convogliare tutti i gli elementi con la stessa chiave nel medesimo reducer e quindi è possibile effettuate la somma facilmente. Il risultato parziale ha come chiave quella composta nella fase di map e come valore il numero totale di casi;
- il secondo mapper crea, con i valori letti dal file di output intermedio, una chiave composta da District e il numero totale di casi e come valore inserisce il campo Description;
- attraverso l'utilizzo di un partizionatore e di un comparatore per la chiave composta, gli elementi vengono ordinati prima per distretto di appartenenza e poi, per ognuno di essi, per numero decrescente di casi totali;
- al secondo reducer gli elementi arrivano già ordinati e quindi è possibile emettere il risultato finale nella forma che si ritiene più conforme, in questo caso: "District, Description e Numero totale di casi."

Al fine di migliorare i tempi di esecuzione, per il primo job è stata utilizzata come combiner la classe definita come reducer, mentre nel secondo, come partitioner si è realizzata una semplice classe per dividere gli item in base ad un codice hash che viene generato e come sortComparator la classe per il confronto della chiave composta. Utilizzando questi accorgimenti il tempo di esecuzione del task è di circa 2 minuti.

Per questo job, come detto in precedenza, sono state provare due versioni. La seconda comprende l'utilizzo di due tabelle e di effettuare il conteggio iniziale per IUCR e non per Description che viene considerata solo in un secondo momento. Questo esperimento ha l'obiettivo di valutare se le performance possano migliorare eseguendo le operazioni su un campo più corto come il codice rispetto alla descrizione. L'implementazione di questa modalità ha richiesto l'introduzione di un terza fase di MapReduce che effettua il join fra le due tabelle mentre le fasi restanti rimango per lo più inalterate. Per effettuare il join si ha bisogno di due mapper, uno che prenda i dati necessari dal file Crimes-2001-to-present-without-description.csv e uno che legga da Chicago-Police-Department-Illinois-Uniform-Crime-Reporting-IUCR.csv, descritti precedentemente. La chiave per i due mapper deve essere al chiave di join e quindi lo IUCR. Nella fase di reducer si dividono i valori che posseggono la stessa chiave in base al file di provenienza, questa operazione è effettuata semplicemente mettendo un prefisso durante il mapping, e viene infine creato il risultato voluto.
Con l'inserimento di una terza fase di MapReduce, le performance del job calano e i tempi di esecuzione passano a circa 3 minuti, 2.45 minuti. Anche questo risultato non deve sorprendere poichè l'introduzione del join rende necessario un ulteriore file intermedio e la lettura da due file di input. Questo provoca un rallentamento per i continui accessi su disco tipici di questo paradigma. Quindi si può concludere che il vantaggio che si può ottenere elaborando un campo più corto viene annullato dall'overhead per la gestione di un ulteriore job. Di seguito si riporta il comando per l'esecuzione del job con il join contenuto nella classe CrimesCountDistrictJoin:

```
hadoop jar BDE-mr-Ventani-Venturi.jar CrimesCountDistrictJoin
progetto_ventani/2/Crimes-2001-to-present-without-description.csv
progetto_ventani/output-conteggio
progetto_ventani/2/Chicago_Police_Department_Illinois\\
_Uniform_Crime_Reporting_IUCR.csv
progetto_ventani/output-join
progetto_ventani/output-count-join
```

L'output è accessibile al percorso `user/aventani/progetto_ventani/output-count-join`

I file con i log di esecuzione di YARN, per le due versioni, possono essere analizzati con i seguenti comandi:

- per l'elaborazione senza join:

```
yarn logs -applicationId application_1583679666662_3991
yarn logs -applicationId application_1583679666662_3992
```

- per l'elaborazione con join:

```
yarn logs -applicationId application_1583679666662_3993
yarn logs -applicationId application_1583679666662_3994
yarn logs -applicationId application_1583679666662_3995
```

Le risorse utilizzate sono state riportate nei file count_crimes_no_join e count_crimes_join presenti nella cartella mapreducelog del progetto.

3.1.2 Spark

3.1.3 Spark

L'implementazione di questo job in Spark è stata eseguita da Simone Venturi. Il codice è contenuto nei file CountCrimesWithJoin e CountCrimesWithoutJoin i quali possono essere eseguiti tramite i comandi:

- spark2-submit --class CountCrimesWithoutJoin bigdata/BDE-spark-Ventani-Venturi.jar

  utilizzando come input di deafult: /user/sventuri/Crimes-2001-to-present-nocomma.csv e come outuput: /user/sventuri/Count-Crimes-2001-to-Present altrimenti si possono specificare input e output aggiungendoli come paramentri, ad esempio:

  ```
  spark2-submit --class CountCrimesWithoutJoin
  bigdata/BDE-spark-Ventani-Venturi.jar
  "/user/sventuri/Crimes-2001-to-present-nocomma.csv"
  "/user/sventuri/Count-Crimes-2001-to-Present"
  ```

  L'input è costituito da un file contenente l'intera tabella dei crimini senza alcuna elaborazione

- spark2-submit --class CountCrimesWithJoin bigdata/BDE-spark-Ventani-Venturi.jar

  utilizzando come input di deafult: /user/sventuri/Crimes-2001-to-present-withoutDescription.csv /user/sventuri/Chicago-Police-Department-Illinois-Uniform-Crime-Reporting-IUCR-Codes-nocomma.csv e come output: /user/sventuri/Count-Crimes-2001-to-Present-WithJoin altrimenti si possono specificare input e output tramite parametrizzazione, ovvero:

  ```
  spark2-submit --class CountCrimesWithJoin
  bigdata/BDE-spark-Ventani-Venturi.jar
  "/user/sventuri/Crimes-2001-to-present-withoutDescription.csv"
  "/user/sventuri/Chicago-Police-Department-Illinois-
  Uniform-Crime-Reporting-IUCR-Codes-nocomma.csv"
  "/user/sventuri/Count-Crimes-2001-to-Present-WithJoin"
  ```

L'input è costituito da un file contenente la tabella dei crimini senza le colonne Primary Type e Description e da un file contenente le tipologie di crimine con le descrizioni. L'output invece è costituito da un file csv con simbolo di separazione ";" che contiene le colonne "District","Description" e "Count" ed è accessibile al percorso specificato come terzo parametro, se specificato, oppure al percorso di default /user/sventuri/Count-Crimes-2001-to-Present-WithJoin.

L'elaborazione senza join è composta dalle seguenti fasi:

- nel primo stage, l'input viene letto e vengono conservate solamente le colonne d'interesse in un dataframe. Successivamente su questo viene effettuato un raggruppamento per record simili e su questo viene eseguito un conteggio di linee in modo tale da poter ricavare quanti crimini dello stesso tipo siano stati commessi nel medesimo distretto.

- nel secondo stage viene salvato il risultato in memoria secondaria.

L'elaborazione con join prevede l'aggiunta della lettura della tabella contenente le tipologie di crimine con le relative descrizioni, questa viene joinata alla tabella contenenti le rilevazioni dei crimini di Chigago con un left outer join in fase finale poco prima di salvare il risultato su memoria secondaria. Le elaborazioni impiegano: 1.6 minuti (la versione senza join) e 1.8 minuti (la versione con join). Al fine di migliorare le performance, sono stati eseguiti questi jobs con diversi parametri:

- numero di executors: default, 20, 40;

- numero di CPU: 6, 20.

Inoltre, al fine di migliorare le performance, è stata testata anche una versione di programma con il join nella quale la tabella con le tipologie di crimine (ha dimensioni notevolmente ridotte) era conservata come una variabile broadcast. Questa versione, dalla quale mi aspettavo una performance migliore, ha deluso le aspettative poiché ha richiesto 2 minuti anziché 1.8 della versione classica con join.

Per visualizzare i dettagli dell'esecuzione senza join:

```
hdfs dfs -cat /user/spark/spark2ApplicationHistory/
application_1583679666662_4332
```

Per visualizzare i dettagli dell'esecuzione con join:

```
hdfs dfs -cat /user/spark/spark2ApplicationHistory/
application_1583679666662_4331
```

3.2 Job #2: conteggio della percentuale di crimini con arresti

Lo scopo del secondo job è il calcolo della percentuale dei crimini in cui è stato effettuato un arresto rispetto al totale. In questo caso i crimini sono raggruppati per tipologia e per anno di registrazione. Il risultato finale in questo caso è:

```
2001;THEFT/RECOVERY: TRUCK,BUS,MHOME;19.0
2001;ARMED: OTHER FIREARM;16.0
...
2002;FRAUD OR CONFIDENCE GAME;14.0
2002;THEFT/RECOVERY: CYCLE, SCOOTER, BIKE W-VIN;73.0
...
```

Dove il primo numero rappresenta ovviamente l'anno di avvenimento del crimine, il secondo la descrizione, campo Description, il terzo una percentuale calcolata come $^{\text{numero totale crimini del gruppo con campo Arrest == True}}/_{\text{numero totale crimini del gruppo}}$

Il file eseguibile per questo job, una volta compilato con gradlew, è nel jar BDE-mr-Ventani-Venturi.jar, per MapReduce, mentre in BDE-spark-Ventani-Venturi.jar per la versione in spark.

3.2.1 MapReduce

L'implementazione di questo job in MapReduce è stata eseguita da Simone Venturi. Il codice prodotto col paradigma MapReduce, presente nella classe PercentageArrested, può essere eseguito, dopo essere stato compilato, con il seguente comando:

```
hadoop jar BDE-mr-Ventani-Venturi.jar
second_job.ArrestedPercentage
Crimes-2001-to-present-nocomma.csv
/user/sventuri/crimespercentageordered/first-output
/user/sventuri/crimespercentageordered/partitions
/user/sventuri/crimespercentageordered/output
```

I parametri necessari all'esecuzione sono quattro: il file di input Crimes-2001-to-present-nocomma.csv che, come detto in precedenza, rappresenta la tabella completa dei crimini dal 2001 ad oggi della città ci Chicago; i percorsi che costituiscono i parametri 2, 3 e 4 del comando sono i due file di output e il file per le partizioni: il primo contiene il risultato intermedio della percentuale di arresti di un determinato crimine per ogni anno, il secondo contiene la codifica delle partizioni utili per l'ordinamento, mentre il terzo è il risultato finale del job. Per ottenere il risultato voluto, si è deciso di dividere la computazione in due job di MapReduce. Nella prima si effettua il conteggio della percentuale di arresti per crimine per ogni anno, successivamente, nella seconda si ordina il risultato ottenuto in ordine crescente per anno e si sistema l'output. Di seguito una breve descrizione le fasi principali:

- dopo il primo mapper il valore della chiave è costituito dalla coppia di valori Year, IUCR mentre il valore è il boolean che contiene l'informazione relativa all'arresto;
- nel primo reducer, viene eseguito il calcolo della percentuale. Il risultato parziale ha come chiave quella composta nella fase di map e come valore la percentuale di arresti;
- il secondo mapper crea, con i valori letti dal file di output intermedio, una chiave composta da Year e come valore inserisce il campo IUCR e percentuale di arresti.
- attraverso l'utilizzo di un partizionatore, il TotalOrderPartitioner, gli elementi vengono suddivisi ordinati per la chiave;
- al secondo reducer gli elementi arrivano già ordinati e quindi è possibile emettere il risultato finale nella forma che si ritiene più conforme, in questo caso: "Year, IUCR e percentuale di arresti".

Al fine di migliorare i tempi di esecuzione, nel secondo job, è stato utilizzato come partitioner il TotalOrderPartitioner in modo tale da poter utilizzare più reducer anziché far eseguire tutta la computazione ad un recuder solo. Utilizzando questi accorgimenti il tempo di esecuzione del task è inferiore a 2 minuti. Per visualizzare il log, eseguire:

```
yarn logs -applicationId application_1583679666662_4333
```

3.2.2 Spark

L'implementazione di questo job in Spark è stata eseguita da Alessia Ventani e il codice è contenuto nel file PercentageArrestedCrimes. Il comando per provare ad eseguire l'elaborazione è:

```
spark2-submit --class PercentageArrestedCrimes
BDE-spark-Ventani-Venturi.jar
"/user/aventani/progetto_ventani/2/Crimes_-_2001_to_present.csv"
"/user/aventani/progetto_ventani/percentage-count"
```

Dal comando precedente si può notare come questo job abbia un solo file di input, rappresentato dal primo parametro, e un solo file di output. L'input è costituito dal file contenente l'intera tabella dei crimini senza alcuna elaborazione. L'output è costituito da un file di formato csv, con simbolo di separazione ";" che contiene le colonne "Year", "Description" e "Percentage crimes with arrest" mentre il risultato è accessibile al percorso `user/aventani/progetto_ventani/percentage-count`

Data la struttura del file di input si è deciso di caricare i dati con il DataFrame che per sua natura si adatta ad un tipo di dato contenuto in una tabella e l'elaborazione è avvenuta con l'api di spark sql. L'elaborazione di compone delle seguenti fasi:

- nel primo stage i dati vengono caricati e vengono selezionate solo le colonne di interesse per questa operazione. I dati poi vengono raggruppati per "Year", "IUCR" e "Description";
- nel secondo stage viene sfruttata l'API di spark sql per poter effettuare una operazione di aggregazione che permette di calcolare la percentuale di casi di crimini con arresti rispetto ai totali per ogni raggruppamento effettuato;
- come ultimo passaggio i dati vengono ordinati in ordine crescente per anno di registrazione e si scrive il risultato di output terminando la computazione.

Il tempo di esecuzione di questa operazione è di circa 1.40 minuti. L'ottimizzazione più grossa che è stata effettuata è quella sul codice stesso. Infatti, in una prima versione, si è optato per la creazione di due data frame distinti per il calcolo della percentuale e in questa maniera si rendeva necessaria un'operazione di join che aumenta di molto i tempi. Con l'utilizzo delle operazioni di aggregazione si è rilevata una grossa diminuzione del tempo e il codice risulta più compatto e leggibile.Inoltre si sono provati i seguenti parametri:

- numero di executors: default, 40 e 20;
- numero di CPU: 6 e 20.

Dalle prove effettuate, si è potuto osservare che il tempo di esecuzione effettivo non varia di molto: nel caso migliore con 20 executors il tempo cala fino a 1.20 secondi: questo risultato è spiegabile poichè il volume di dati da elaborare non è così elevato e quindi anche ottimizzando al massimo le risorse le performance non cambiano sensibilmente. Aumentando il numero di executors invece il tempo inizia a salire: l'overhead per il coordinamento dei dati supera l'efficacia di poter parallelizzare l'elaborazione stessa. Con un database ancora più esteso probabilmente questi accorgimenti possono essere molto utili.

Il log di questa elaborazione può essere consultato eseguendo il seguente comando:

```
hdfs dfs -cat /user/spark/spark2ApplicationHistory/
application_1583679666662_3996
```

4 Descrizione del risultato

Una volta eseguiti i due job si è deciso di utilizzare Tableau per visualizzare i risultati. I file sono contenuti nella cartella output di GitHub.
Dopo aver scaricato in locale e in un unico file csv i risultati dei job, sono stati aperti in Tableau ed è stato realizzato un grafico per ognuno. In particolare per il primo job il risultato è visibile nel grafico count_crime.png e per il secondo in percentage.png.

Da questa operazione è stato possibile osservare alcuni andamenti dei dati nei due job interessanti. In particolare:

- nel primo job: il conteggio di tipi specifici di crimini è ovviamente molto più alto di tutti gli altri, come per esempio i crimini con multa minore di 500 dollari che ovviamente risultano più frequenti;
- nel secondo job: alcune categorie di crimini hanno un'alta percentuale di casi con arresti mentre altri pari a zero. Questo è spiegabile dalla tipologia di caso. Per esempio i crimini con 300 dollari, che prevedono una ammenda e non una incarcerazione, hanno una percentuale pari a zero mentre altri invece hanno una percentuale molto alta, come il crimine di possesso illegale di armi con attacco ad una persona.

Visualizzare con grafici i dati ottenuto è stato utile al fine di avere un colpo d'occhio globale e rilevare delle peculiarità che aprono la porta a possibili ulteriori elaborazioni.