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

Alessia Ventani - Mat. 901809
Simone Venturi - Mat.

3 luglio 2020

Indice

1 Teachers' notes **3**

2 Introduzione **4**
2.1 Descrizione del dataset 4
2.1.1 Descrizione dei file 4

3 Preparazione dei dati **5**

4 I Job **6**
4.1 Job #1: conteggio dei crimini per distretto 6
4.1.1 MapReduce 7
4.1.2 Spark 9
4.2 Job #2: conteggio della percentuale di crimini con arresti 9
4.2.1 MapReduce 10
4.2.2 Spark 10

5 Descrizione del risultato **11**

1 Teachers' notes

The goal of the project is to assess the students' skills in writing jobs of low/medium complexity and to correctly reason about the jobs' performances. The projects must be agreed with the teachers (do not start without explicit consent) and it consists of:

- finding a complex-enough dataset: about 1GB, possibly consisting of more tables;
- loading the dataset on HDFS/Hive;
- implement an analytical job in both MapReduce and Spark;
- writing a short report to describe it.

To deliver the project, each group must **send by email to both teacher the link to their repository**. The repository must:

- be created from the individual assignment available on the Github classroom;
- contain a report folder with the PDF of the final report (based on this template). The report can be written in either Italian/English and Latex/Word at your discretion. Be concise and go straight to the point: an excessively verbose report is a waste of time for you and for the teachers;
- contain a README.md file with the instruction to run the jobs. Indeed, the teachers must be able to clone the repository and run the jobs from their accounts on the cluster. This means that the dataset must be accessible on HDFS/Hive and the code must compile and run correctly. Please make the jobs repeatable (i.e., the job checks and possibly deletes old data to avoid errors when re-running the code).

This guide is based on the "MapReduce+Spark" kind of project. However, we remind that a different kind of project may be agreed upon.

The evaluation will be based on the following.

- Compliance of the jobs with the agreed upon specifications.
- Compliance of the report with this guide.
- Job correctness.
- Correct reasoning about optimizations.

Appreciated aspects.

- Code cleanliness and comments.
- Further considerations in terms of job scalability and extensibility.

2 Introduzione

2.1 Descrizione del dataset

Please provide:

- A brief description of the dataset.
- The link to the website publishing the dataset (e.g., `https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page`).
- Direct links to the downloaded files, especially if more than one files are available in the previous link (e.g., `https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripdata_2017-01.csv`).

Per questo elaborato si è deciso di utilizzare gli open data che la città di Chicagoha 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, della città.
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-c7ck-438e`

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

2.1.1 Descrizione dei file

For each file, briefly indicated the available data and the fields used for the analyses; examples are welcome.

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 di dati riportati è:
IUCR Description District Arrest Year
0460 SIMPLE 006 False 2020

3 Preparazione dei dati

Please provide:

- The paths to each file on HDFS and/or its corresponding location in Hive (database and table); consider relying on the structured data lake organization.
- A subsection with details on the pre-processing of the data (only necessary if the data is dirty and/or it contains a significant amount of useless information).

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 dei confronti 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.

- The paths to each file on HDFS and/or its corresponding location in Hive (database and table); consider relying on the structured data lake organization.
- A subsection with details on the pre-processing of the data (only necessary if the data is dirty and/or it contains a significant amount of useless information).

4 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.

4.1 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
...
```

Dove 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.

4.1.1 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
```

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 ed e l'ordinamento degli item precedenti. 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 scrittura e la lettura su diversi file. 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
```

4.1.2 Spark

L'implementazione di questo job in MapReduce è stata eseguita da Simone Venturi. Please provide:

- Input and output files/tables.
- Execution time and amount of resources.
- Direct links to the application's history on YARN (e.g., `http://isi-vclust0.csr.unibo.it:18088/history/application_15...`).
- Description of the implementation. A schematic and concise discussion is preferrable to a verbose narrative. Focus on how the data is manipulated in the job (e.g., what do keys and values represent across the different stages, what operations are carried out).
- Performance considerations with respect the (potentially) carried out optimizations, e.g., in terms of:
  - allocated resources and tasks;
  - enforced partitioning;
  - data caching;
  - combiner usage;
  - broadcast variables usage;
  - any other kind of optimization.
- Short extract of the output and discussion (i.e., whether there is any relevant insight obtained).

4.2 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 $\frac{\text{numero totale crimini del gruppo con campo Arrest == True}}{\text{numero totale crimini del gruppo}}$.

Il file eseguibile per questo job, una volta compilato con gradlew, è nel jar BDE-spark-Ventani-Venturi.jar.

4.2.1 MapReduce

L'implementazione di questo job in MapReduce è stata eseguita da Simone Venturi. Please provide:

- Performance considerations with respect the (potentially) carried out optimizations, e.g., in terms of:
  - allocated resources and tasks;
  - enforced partitioning;
  - data caching;
  - combiner usage;
  - broadcast variables usage;
  - any other kind of optimization.
- Short extract of the output and discussion (i.e., whether there is any relevant insight obtained).

4.2.2 Spark

L'implementazione di questo job in MapReduce 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_crimes_with_arrest_by_year"
```

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".

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.

5 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