BIG DATA
Relazione di progetto

Michele Proverbio
michele.proverbio@studio.unibo.it

Dataset

Il dataset utilizzato è stato preso dalla piattaforma Kaggle. Si tratta di uno studio sulla criminalità londinese condotto dal 2008 al 2016 dalla città di Londra; e conta 13 milioni di record per uno spazio totale di 910 MB.

Struttura e considerazioni sugli attributi

I record sono definiti da 7 attributi:

- *lsoa_code*: codice Lower Super Output Area. Identifica i quartieri di Londra.
- *borough*: Nome comune del quartiere.
- *major_category*: Categoria di alto livello del crimine commesso.
- *minor_category*: Sotto categoria del crimine commesso.
- *value*: occorrenze del crimine riscontrare nel mese.
- *year*: anno [2008 - 2016]
- *month*: mese [1 - 12]

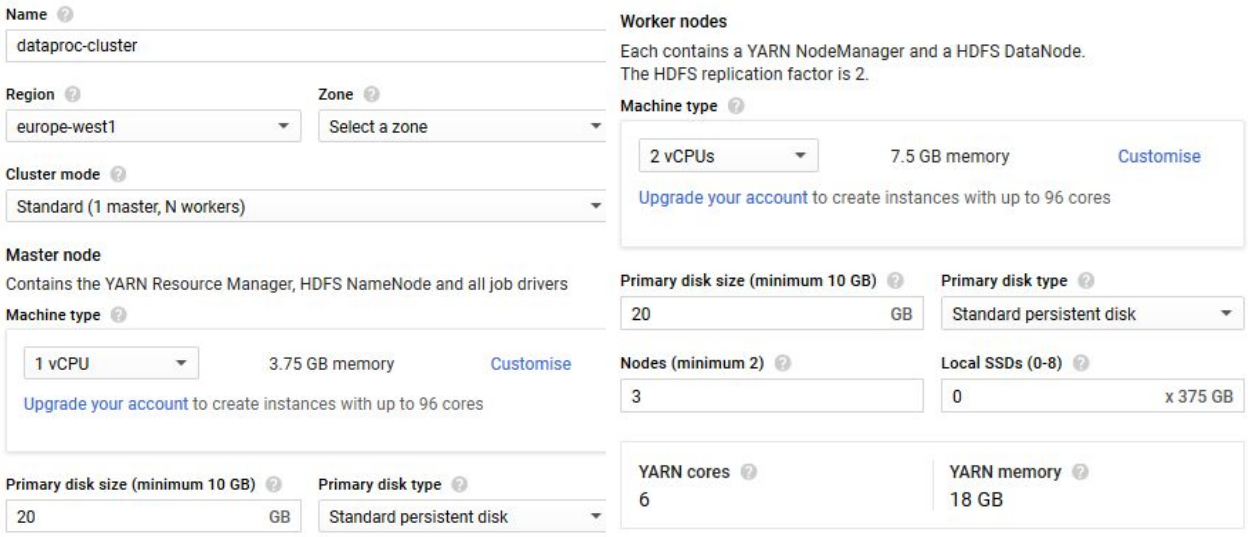
I Primi quattro attributi sono categorici testuali, mentre gli ultimi tre sono attributi numerici.

Ci sono record con valore *value* nullo (0) se non sono stati commessi crimini di quel tipo nel mese *month* e nel quartiere *borough.*

Infrastruttura cluster

Per eseguire i job è stato creato un cluster di 4 nodi su Google Cloud. 1 master node con 1 vCPU e 3.75GB di memoria e 3 worker nodes con 2 vCPU e 7.5GB di memoria.

Google mette a disposizione il prodotto *Dataproc* che permette la creazione di cluster con macchine già preconfigurate con tutto lo stack hadoop già installato.



Dataproc fornisce accessi ssh a tutte le macchine e interfacce web per la sottomissione e il monitoraggio dei job, oltre che ad un monitoraggio semplificato delle risorse delle macchine.

Problemi riscontrati

In due occasioni i worker node sono stati sospesi da Google per apparente uso illecito delle macchine. In particolare hanno rilevato un possibile utilizzo di software di crypto mining.

La prima volta è bastato segnalare il false positive a Google per sbloccare le macchine, ma la seconda sospensione è stata invece permanente. È stato quindi necessario ricreare e riconfigurare il cluster.

Job 1: lista ordinata dei quartieri di londra per occorrenze di un determinato crimine negli ultimi 5 anni

MapReduce

Il job mapreduce si svolge in due fasi. La prima fase filtra i record per anno e crimine nei mapper e aggrega i dati sommando le occorrenze del crimine nei reducer.

La seconda fase utilizza il *shuffle and sort* di mapreduce per ordinare i quartieri per occorrenze utilizzando la classe base *Reducer* offerta da hadoop.

Spark

Le prime trasformazioni al RDD sono operazioni di filtro e selezione per ridurre subito le dimensioni dei dati su cui lavorare. Successivamente viene fatta aggregazione sulla colonna *occurrencies* applicando l'operatore di somma. Infine, il RDD viene ordinato in modo decrescente sulla nuova dimensione aggregata.

Job 2: per ogni anno determinare i 3 quartieri con la media di crimini al giorno più alta

MapReduce

Il job è composto di due fasi: la prima crea le chiavi composite (quartiere e anno) nel mapper per poi usarle come aggregatore per la somma delle occorrenze. In questo modo riesco ad ottenere la somma dei crimini mantenendo la granularità sia sull'anno che sul quartiere.

Il mapper della seconda fase passa invariati i dati al reducer utilizzando l'anno come nuova chiave.

Nel reducer ho scelto di utilizzare gli stream di java 8 per ordinare i quartieri per numero di crimini e prenderne i primi 3.

Al termine del job ogni reducer avrà scritto un file parziale con i 3 quartieri con più crimini in quell'anno.

Spark

Per prima cosa viene creata una *window* sui dati con partizione sull'anno e ordinamento sulle occorrenze dei crimini. La *window* servirà a creare una colonna *rank* per poter filtrare successivamente solo i primi 3 quartieri.

La prima operazione sul RDD è il group-by sulle dimensioni *anno* e *quartiere* aggregando sulla somma.

Successivamente viene creata un'altra colonna *rank* utilizzando la window creata in precedenza . Il rank viene utilizzato per filtrare solo i primi 3 quartieri per ogni anno (rank <= 3).

Infine calcolo la media annua dei crimini dividendo il valore aggregato per 365 e salvo su csv.

Considerazioni sulle prestazioni

I tempi di calcolo ottenuti dalla gestione dei job in cloud di dataproc sono riportati nella seguente tabella:

	MapReduce	Spark	Differenza %
job1	1m33s	1m30s	~1%
job2	1m37s	1m17s	20%

Nel caso del primo job la differenza di prestazione è molto poco significativa.

Probabilmente i dati rimanenti dalle fasi di filtraggio e aggregazione non erano abbastanza grandi per permettere alla gestione in-memory di spark di impattare in modo significativo sulle prestazioni.

Nel secondo caso invece Spark guadagna un buon margine su MapReduce. Una criticità su cui MapReduce perde tempo è sicuramente il passaggio dei dati dalla prima alla seconda fase. Diversamente dal primo job non c'è una fase di filtraggio, e l'aggregazione fatta nel reducer è più granulare rispetto a quella del job1. Questo fa sì che la dimensione dei dati intermedi dia a Spark un importante edge su MapReduce.

Un altro punto critico è il windowing. Spark ci permette di utilizzarlo nativamente, mentre per MapReduce ho dovuto utilizzare gli stream di java 8.

Considerazioni sulle tecnologie

Spark permette di ridurre considerevolmente la complessità degli algoritmi, riuscendo ad avere un codice molto più snello ed espressivo rispetto a MapReduce.

Ho scelto Python per Spark per testare un ambiente diverso da Java anche per questioni lavorative future.

Continuous Integration and Deployment. Gitlab e gcloud

Ho deciso di utilizzare la potenza della gestione integrata di Dataproc mettendo in piedi una pipeline per la compilazione e la sottomissione dei job automatica.

Quando una modifica viene pushata nel repository git di Gitlab viene lanciata una pipeline di CI/CD previamente configurata. Gli stage della pipeline vengono eseguiti da un *runner* sul master node:

1. Il codice java viene compilato e vengono creati i jar
2. Vengono sottomessi i job Mapreduce e Spark attraverso l'utilizzo delle API Google Cloud
3. Vengono aggregati gli output e resi disponibili al download nella pipeline
4. Cleanup

Un esempio di utilizzo delle API per la sottomissione di un job spark:

```
gcloud dataproc jobs submit pyspark --region europe-west1 --cluster
dataproc-cluster python/job1.py
```

Una volta lanciato il submit, il job sarà visibile in Dataproc con conseguente storicizzazione e gestione di risorse e log.