PROGETTO: *Comparatore di Applicazioni*
ESAME: *Piattaforme per i Big Data*

Alessio Cimino ***a.a. 2024***

Il progetto sviluppato durante questo corso ha come obiettivo fornire agli utenti uno strumento che consenta loro di filtrare le applicazioni specificando parametri in base alla richiesta. Ogni giorno, infatti, vengono create e aggiunte nuove applicazioni con una facilità sempre crescente.

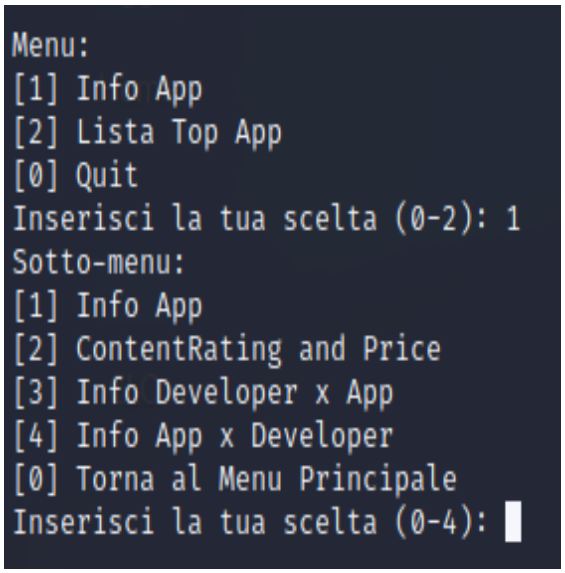
È stato utilizzato un archivio che contiene più di 2,3 milioni di applicazioni sviluppate per il sistema operativo Android e scaricabili attraverso la piattaforma Google Play.

Il file è aggiornato fino a giugno 2021 e rappresenta un insieme di dati di Big Data indispensabile per elaborare una grande quantità di informazioni, come in questo caso specifico.

Questo file contiene, per ogni riga, le seguenti 23 caratteristiche:

- App Name
- App Id
- Category
- Rating
- Rating Count
- Installs
- Minimum Installs
- Maximum Installs
- Free
- Price
- Currency
- Size
- Minimum Android
- Developer Id
- Developer Website
- Developer Email
- Released
- Privacy Policy
- Last Updated
- Content Rating
- Ad Supported
- In app purchases
- Editor Choice

L’interfaccia grafica vuole essere semplice ed intuitiva a causa dell'enorme quantità di dati; perciò ho diviso i metodi in due parti, come mostrato nella figura:



La prima parte dei metodi genera un output direttamente sul monitor, selezionando una delle opzioni nel sotto-menu visualizzato.

La seconda opzione, al contrario, genera, una volta eseguita, un file .csv con un numero di righe specificato dall'utente in un intervallo che va da 1 a 1000. L'utente può specificare direttamente le sotto-classi di interesse, applicando filtri prima di lanciare la query.

Tale decisione è volta a sviluppare in modo ottimale le linee di codice implementate, rendendo più efficiente l'esecuzione ed evitando di utilizzare inutilmente risorse computazionali e spazio di memoria fisica.

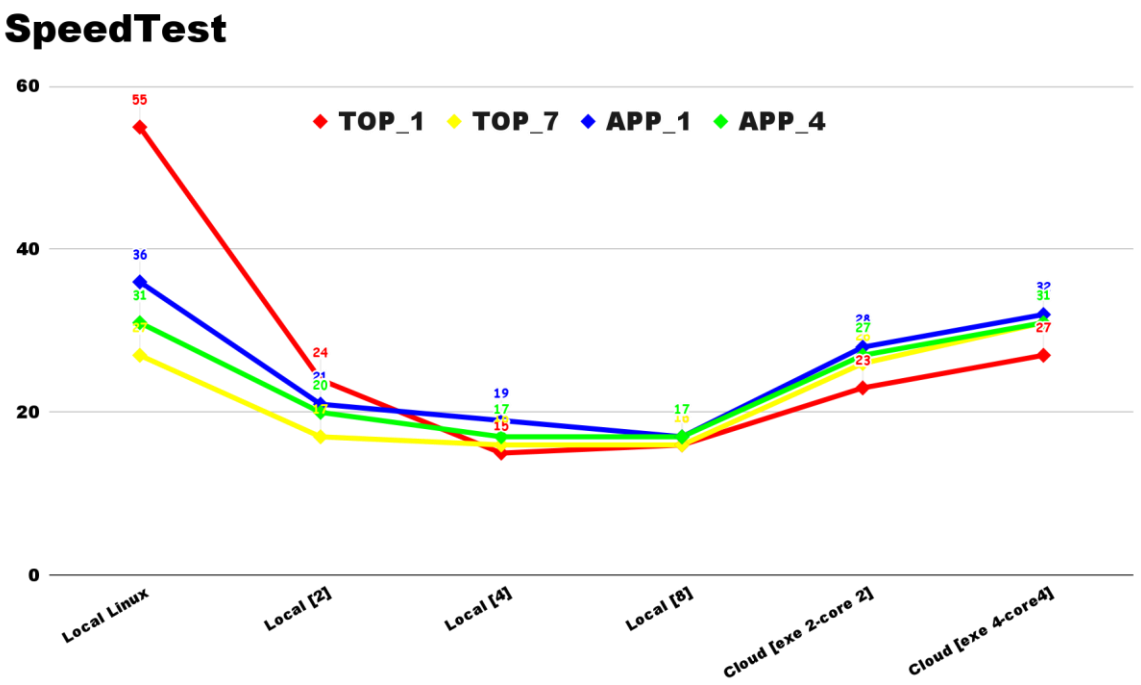
I metodi sono stati inizialmente testati su una macchina virtuale Kali Linux su Oracle installata su un laptop con le seguenti caratteristiche hardware:

- Processore AMD 3020e (2C / 2T, 1.2 / 2.6GHz, 1MB L2 / 4MB L3)
- Memoria RAM da 8GB SO-DIMM DDR4-2400
- Archiviazione SSD da 256GB M.2 2242 PCIe NVMe 3.0x2
- Scheda grafica integrata AMD Radeon Graphics
- Sistema operativo: Windows 10

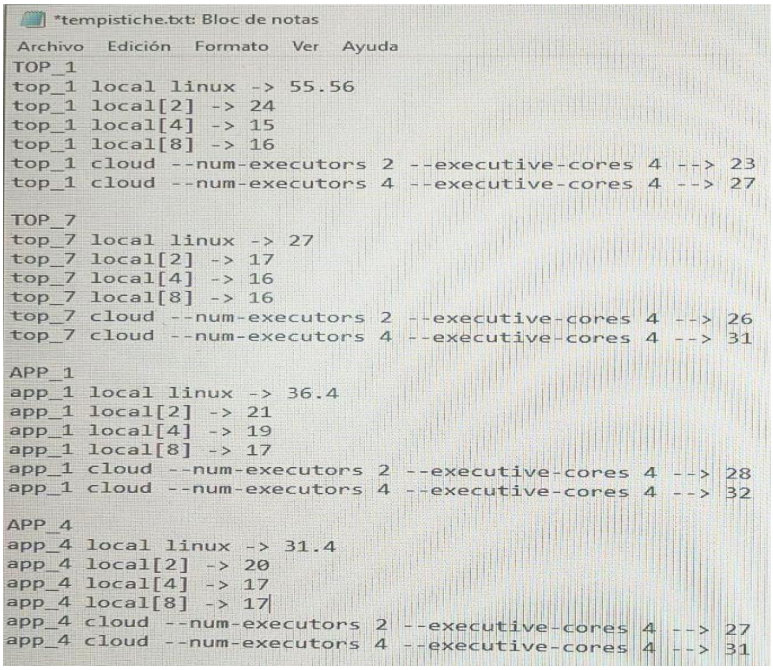
Successivamente, modificando una piccola parte del codice per eliminare la parte interattiva del menu, ho testato i vari metodi in modalità Cloud per confrontarne le prestazioni, ottenendo i grafici qui di seguito rappresentati.

Rendimento:

Tempi di esecuzione - Local Virtual Machine e Google Cloud



Come si può osservare nel grafico presentato, i tempi di esecuzione nella modalità locale diminuiscono significativamente all'aumentare del numero di core utilizzati. Naturalmente, i tempi rappresentati dall'esecuzione in cloud sono più elevati rispetto a quelli locali a causa del tempo necessario per caricare i dati.



*tempistiche.txt: Bloc de notas

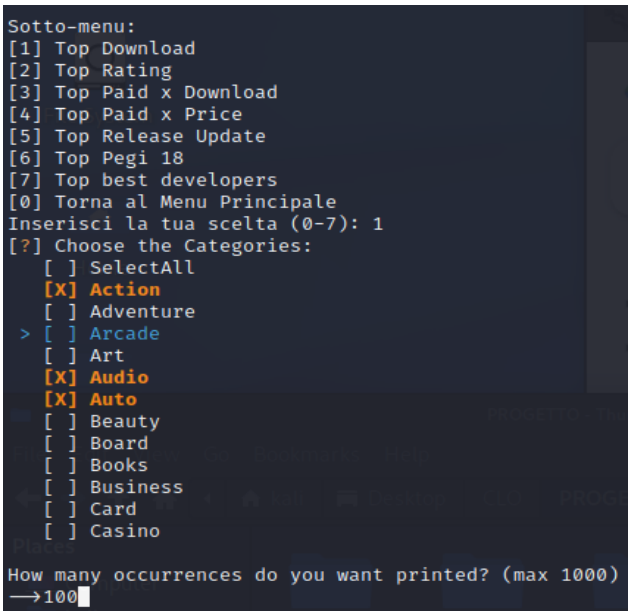
Archivo Edición Formato Ver Ayuda

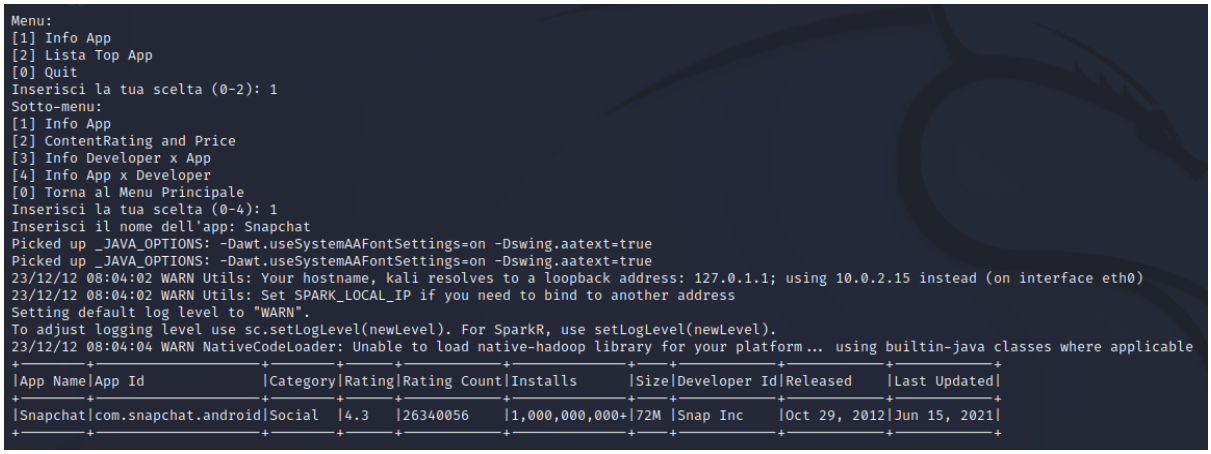
```
TOP_1
top_1 local linux -> 55.56
top_1 local[2] -> 24
top_1 local[4] -> 15
top_1 local[8] -> 16
top_1 cloud --num-executors 2 --executive-cores 4 --> 23
top_1 cloud --num-executors 4 --executive-cores 4 --> 27

TOP_7
top_7 local linux -> 27
top_7 local[2] -> 17
top_7 local[4] -> 16
top_7 local[8] -> 16
top_7 cloud --num-executors 2 --executive-cores 4 --> 26
top_7 cloud --num-executors 4 --executive-cores 4 --> 31

APP_1
app_1 local linux -> 36.4
app_1 local[2] -> 21
app_1 local[4] -> 19
app_1 local[8] -> 17
app_1 cloud --num-executors 2 --executive-cores 4 --> 28
app_1 cloud --num-executors 4 --executive-cores 4 --> 32

APP_4
app_4 local linux -> 31.4
app_4 local[2] -> 20
app_4 local[4] -> 17
app_4 local[8] -> 17
app_4 cloud --num-executors 2 --executive-cores 4 --> 27
app_4 cloud --num-executors 4 --executive-cores 4 --> 31
```

L'infrastruttura utilizzata per sviluppare il progetto include diverse risorse informatiche, come PySpark con molte librerie essenziali per analizzare diversi tipi di dati (stringhe, interi, date, booleani, ecc.), Google Cloud e, naturalmente, Excel per verificare l'accuratezza dei dati estratti.

```python
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, unix_timestamp, datediff, current_date
from pyspark.sql.types import TimestampType

import sys

def generate_top_lastUpdated_csv(listaGeneros, output_path, number_choice):
    spark = SparkSession.builder.appName("ToplastUpdated").getOrCreate()

    file_path = "Google-Playstore.csv"
    df = spark.read.option("header", "true").csv(file_path)

    filtered_df = df.filter(col("Category").isin(listaGeneros))
    filtered_df = filtered_df.filter(col("Last Updated").isNotNull() & (col("Last Updated") ≠ ""))
    date_format = "MMM d, yyyy"
    filtered_df = filtered_df.withColumn("Last Updated", unix_timestamp(col("Last Updated"), date_format).cast(TimestampType()))
    filtered_df = filtered_df.withColumn("DateDifference", datediff(current_date(), col("Last Updated")))
    sorted_df = filtered_df.orderBy("DateDifference")

    top_df = sorted_df.limit(number_choice)
    result_df = top_df.select(
        "App Name",
        "Released",
        col("Last Updated").cast("date").alias("Last Updated"),
        "DateDifference"
    )

    result_df.write.option("header", "true").csv(output_path, mode="overwrite")

    spark.stop()
    print(f"File CSV '{output_path}' generato con successo.")

print("Argomenti di input:", sys.argv)
listaGeneros = sys.argv[1].split(',')
number_choice = int(sys.argv[2])

if not listaGeneros:
    print("Specifica almeno un genere.")
    sys.exit(1)

output_path = "output_top/Top_lastUpdated.csv"
generate_top_lastUpdated_csv(listaGeneros, output_path, number_choice)
```

Il menù utente implementato si basa su una interfaccia semplice ed intuitiva, leggera ed essenziale, come mostrato nella figura qui accanto, nella quale è possibile specificare la categoria di interesse (o più di una) ed il numero di righe output desiderate per un massimo di 1000.





I risultati ottenuti, al di là dei tempi di esecuzione che possono facilmente variare in base alla configurazione hardware utilizzata, mostrano un risultato in linea con le aspettative del progetto presentato per il quale sarà possibile, in un secondo momento, implementare metodi addizionali che si rendano necessari per ottimizzare un sistema di consultazione di questo tipo, aggiornando ovviamente il DataSet utilizzato.