# Progetto Finale Big Data

Davide Di Maria Matricola: 460009
Antonio Martinelli Matricola: 461981

17 Luglio 2017

## 1 Introduzione

Il nostro progetto finale del corso di Big Data 2016/2017 prevede l'implementazione di un sistema di *Polyglot Persistence* con capacità di profilazione automatica, ovvero di un sistema in grado di processare e analizzare dei dati memorizzati su piattaforme differenti e capace di automatizzare,in parte, la rilevazione delle caratteristiche comuni tra due o più *dataset*.
Le tecnologie utilizzate per realizzare la parte della persistenza poliglotta sono state principalmente due:

1. *MongoDB*: uno dei più famosi database *NoSql* orientato ai documenti;
2. *DataLake*: implementato tramite l'utilizzo del file system distribuito di *Hadoop*.

L'implementazione è stata testata sia in locale che su cluster *Amazon* ma con una sottile differenza per quello che riguarda le tecnologie di supporto.
La versione in locale, infatti, utilizza dei container *Docker* mentre quella su cluster fa uso nativo della piattaforma messa a disposizione da Amazon.
Le differenze tra le due, comunque, verranno analizzate in dettaglio nella sezione successiva a questa, in cui verrano spiegate le problematiche tecnologiche che hanno portato a questa scelta.
I dataset utilizzati per il testing del sistema implementato sono:

1. *Global Terrorism Database*: un database open source di attacchi terroristici dal 1970 al 2015, salvato in MongoDB.
2. *World Development Indicators*: raccolta di indicatori sullo sviluppo mondiale raccolti da *World Bank*, dati memorizzati nel Datalake come i successivi;
3. *World Gender Statistics*: raccolta di indicatori specifici divisi per genere su demografia, educazione e altro, offerto anch'esso da *World Bank*;

4. *Correlates of War: World Religions*: un database che riporta a vari livelli di granularità la distribuzione della popolazione in relazione al credo religioso (ovviamente abbiamo usato la grana degli Stati);

5. *World Energy Use 1960-2012*: che riporta i dati sul consumo energetico dei vari stati dal 1960 al 2012;

6. *WorldBank - Gender Statistics*: un dataset di indicatori fornito da *Kapsarc* (https://www.kapsarc.org) molto più ricco del secondo della lista.

Il resto della relazione è organizzato come segue: nella sezione 2 vengono descritte le tecnologie utilizzate per l'implementazione della piattaforma, mostrando e motivando le differenze tra la versione in locale e quella su cluster.
Nella sezione 3 vengono discusse in maniera approfondita le analisi semplici, ovvero quelle eseguite su una singola sorgente dei dati, nel caso specifico *MongoDB*.
Nella sezione 4 vengono esaminate in dettaglio le analisi poliglotte, ovvero quelle che correlano dataset provenienti da sorgenti diverse, e che quindi risultano più complicate vista la necessità di "normalizzare" i dati in un formato condiviso.
Nella sezione 5 si parla del modulo *Profiler* che ha il compito di automatizzare l'integrazione di dataset memorizzati in basi di dati differenti.
Nella sezione 6 vengono riportati i risultati delle analisi implementate durante lo sviluppo, con una disamina dei tempi osservati.
Infine nell'ultima sezione sono riportate le conclusioni sul lavoro svolto e vengono presentati degli spunti per un possibile sviluppo futuro della piattaforma implementata, in particolar modo per quello che riguarda il modulo di profilazione.

# 2 Tecnologie Utilizzate

Le tecnologie impiegate per lo svolgimento delle analisi differiscono, come già precedentemente accennato, tra la versione locale e quella su cluster. La versione locale del progetto utilizza dei container Docker, ed in particolare due immagini presenti nel *Dockerhub*:

1. *Izone/hadoop*: una immagine predefinita contenente Hadoop e le configurazioni per il suo file system. Rappresenta il Datalake nel contesto presentato;
2. *Mongo*: l'immagine ufficiale del database MongoDB per Docker.

L'utilizzo di questi container ha permesso una configurazione immediata dell'ambiente, definita tramite un semplice script, i cui passaggi principali vengono riportati di seguito per una analisi più approfondita.

1. docker run -dit –name=mongoDB
   -p 80:80 -p 27017:27017 mongo

2. docker run -dit –name=hadoop -p 8088:8088
   -p 8042:8042 -p 50070:50070 -p 8888:8888 -p 4040:4040
   izone/hadoop

3. docker cp Jars/mongo-spark-connector_2.10-2.0.0.jar
   hadoop:/mongo-spark-connector_2.10-2.0.0.jar

4. docker cp Jars/mongo-java-driver-3.4.2.jar
   hadoop:/mongo-java-driver-3.4.2.jar

I primi due comandi, riportati nel riquadro sovrastante, hanno il compito di mandare in esecuzione i due container in modalità *background* in modo da far partire i servizi richiesti per le analisi, ovvero Hadoop e MongoDB, senza interrompere l'utente.
Una menzione particolare va riservata alle opzioni *-p* che hanno il compito di impostare la connettività dei container, provvedendo all'esposizione delle porte specificate per un accesso semplice ai servizi residenti nei docker.
Gli ultimi due comandi hanno invece il compito di copiare all'interno del Docker contenente il Datalake i jar richiesti per la comunicazione tra il *framework Spark* e il database MongoDB.
Lo script si occupa anche di altre opzioni basilari come quelle di caricare nei container i dataset necessari per le analisi e di scaricare ed impostare il framework Spark all'interno del Datalake.

L'ordine di accensione dei due container è importante, infatti la piattaforma Docker ha delle convenzioni nell'assegnazione degli *IP* ai container che vanno rispettati nel caso si volesse automatizzare l'impostazione dell'ambiente di sviluppo, come in questo caso.
La piattaforma utilizzata per l'esecuzione su cluster, come già accennato, è stata quella messa a disposizione dai servizi web di Amazon.
Questo ambiente di sviluppo non ha, al momento della stesura di questa relazione, una integrazione nativa con Docker, non permettendo, perlomeno in maniera agevole, la comunicazione dall'esterno con servizi messi in esecuzione all'interno di questi container.
Questa problematica ha condotto all'utilizzo nativo delle due componenti, Datalake e database MongoDB, all'interno delle macchine offerte dalla piattaforma.
La configurazione del database orientato ai documenti è stata quindi ottenuta attraverso il caricamento di uno speciale file *mongodb-ord-3.4.repo* che ne ha permesso l'installazione automatica tramite il gestore dei pacchetti offerto da Amazon.

# 3 Analisi Base

In questa sezione vengono approfondite le analisi su singolo dataset implementate nel progetto, ovvero quelle considerazioni effettuate in relazione ai dati degli attacchi terroristici memorizzati in MongoDB.
Questi esami sono stati fondamentali per il test dell'interoperabilità tra il framework Spark e MongoDB, facilitato dall'utilizzo di connettori messi a disposizione dal produttore del database stesso.

## 3.1 Most Attacked Countries

La prima analisi implementata persiste in MongoDB l'elenco delle nazioni che hanno subito più attacchi terroristici dal 1970 al 2015.
I dati vengono prelevati dal database attraverso l'utilizzo del connettore Spark ufficiale, che permette un accesso alla sorgente dei dati immediato attraverso la specifica del *JavaSparkContext*, così come riportato di seguito.

```
SparkSession spark = SparkSession.builder()
.appName("MostAttackedCountries")
.config("spark.mongodb.input.uri",
"mongodb://172.17.0.2:27017/dbTerr.attacks")
.config("spark.mongodb.output.uri",
"mongodb://172.17.0.2:27017/dbTerr.mostAttackedCountries")
.getOrCreate();
JavaSparkContext jsc =
new JavaSparkContext(spark.sparkContext());
```

I parametri passati attraverso l'opzione *config* indicano la sorgente dei dati da cui prelevare i dati e la locazione dove salvare l'output (nome del database e della collezione per entrambe le configurazioni) del processamento.
L'URI inserito fa riferimento al servizio di MongoDB in esecuzione nel container Docker precedentemente descritto.
La manipolazione dei dati viene effettuata tramite l'utilizzo delle funzioni *lambda* di Java 8, base del framework Spark.
Viene riportato, di seguito, lo pseudocodice della procedura implementata per lo svolgimento di questa analisi.

```
data.map(country,1)
data.aggregateByKey()
data.map(numOfAttacks,country)
data.sortByKey(descendent)
data.groupbyKey()
```

La fase di *Map* del processamento è molto semplice ed ha il compito di creare

una struttura intermedia contenente il nome della nazione attaccata ed un 1 che verrà poi utilizzato nella prima fase di Reduce.
Completata questa struttura avviene una fase di aggregazione che ha il compito di calcolare il numero di attacchi subiti per ogni nazione attraverso la funzione *aggregateBykey* di Java.
Infine avviene una seconda fase di Map che scambia la posizione dei due campi in modo da poter stilare una classifica delle nazioni più attaccate (l'ordinamento è molto più performante se eseguito sulla chiave della struttura) attraverso la chiamata della funzione *sortByKey*.
L'ultima operazione eseguita è quella di raggruppare i dati per chiave nel caso in cui ci fossero più nazioni con lo stesso numero di attacchi e rendere l'output privo di ripetizioni.
Il salvataggio del risultato all'interno di MongoDB necessita di una operazione intermedia, ovvero quella di convertire i dati restituiti dal framework Spark in un formato adatto alla memorizzazione nel database.
Questo processamento viene eseguito dal metodo *mapToMongo()*, che si occupa di estrarre le informazioni da ogni tupla del *Resilient Distributed Dataset*, o *RDD*, di Spark e convertirla nel formato della classe *Document* di MongoDB.

## 3.2 Claimed Attacks

Questa analisi restituisce il gruppo terroristico che ha rivendicato più attacchi nel periodo storico messo a disposizione dal dataset.
L'implementazione è molto simile a quella dell'analisi precedente e quindi non viene riportata.
L'unico passaggio degno di nota è quello che riguarda l'utilizzo di un filtro atto a limitare il conteggio degli attacchi terroristici senza rivendicazione diretta, in modo tale da avere una lista non contaminata da calcoli su attacchi anonimi.
Come per il processamento precedente c'è stata la necessità di un *mapping* per avere una struttura adeguata per la persistenza nella base di dati.

## 3.3 Attaccks Per Year

Questa indagine restituisce una classifica degli attacchi effettuati per anno.
Il procedimento segue quello illustrato nel dettaglio nella sezione 2.1 e non è quindi riportato.
Una considerazione importante, però, è collegata alla necessità di avere, per ogni classe, un metodo scritto *ad hoc* per il mapping dei dati su MongoDB, fattore dovuto alla grande varietà di analisi che si possono eseguire su questi dataset.

# 4 Analisi Poliglotte

In questa sezione vengono illustrate in dettaglio le analisi poliglotte sviluppate nel corso del progetto.
Il principale ostacolo che si è dovuto affrontare durante questa tipologia di analisi è stato quello di effettuare il join tra due strutture dati completamente differenti.
Nella sezione della profilazione verrà poi evidenziata con maggior dettaglio la volontà di rendere questa procedura automatizzata e quindi di favorire l'inserimento nel Datalake di sorgenti dati correlate a quanto presente nel database principale, ovvero quello rappresentato dall'istanza di MongoDB.

## 4.1 Attacks-Defense Expenditure

L'analisi, qui descritta in dettaglio, ha il compito di mettere in correlazione il numero di attacchi subiti da una nazione con la spesa media investita per il dipartimento della Difesa.
Come accennato precedentemente, i due dataset utilizzati per queste analisi sono totalmente indipendenti e quindi rappresentano un buon banco di test per lo sviluppo di analisi poliglotte, essendo infatti, come avviene nei casi reali, non privi di una grande quantità di dati eterogenei da gestire ed esaminare.
Il caricamento dei dati dalle due sorgenti avviene attraverso la configurazione dello *SparkJavaContext* per quello che riguarda MongoDB mentre i *path* di input e output per il Datalake vengono passati come parametri al programma.

```
if (args.length < 2) {
System.err.println("File path or Output location not found!");
System.exit(1);
}
AttacksDefenseExpenditure att =
new AttacksDefenseExpenditure(args[0]);
SparkSession spark = SparkSession.builder()
.appName("AttacksDefenseExpenditure")
.config("spark.mongodb.input.uri",
"mongodb://172.17.0.2:27017/dbTerr.attacks")
.config("spark.mongodb.output.uri",
"mongodb://172.17.0.2:27017/dbTerr.attacksDefenseExpenditure")
.getOrCreate();
JavaSparkContext jsc =
new JavaSparkContext(spark.sparkContext());
```

Come riportato nel codice viene sollevato un errore se non vengono passati al programma le locazioni di input e output per il Datalake.
Per una analisi più dettagliata vengono di seguito riportati gli pseudocodici delle procedure di join e dell'analisi vera e propria implementata dal programma.

```
dataFromLake.filter(defenseCode = present)
dataFromLake.map(country, restOfTheLine)
dataFromMongo.map(doc.getCountry(), doc.values())
join = dataFromLake.join(dataFromMongo)
join.map({country,defenseExpense},ListOfAttacks)
```

Come risulta dallo pseudocodice, la procedura di join necessita di un mapping delle due sorgenti di dati in un formato condiviso.
Per quanto riguarda il Datalake, questo formato viene ottenuto filtrando i record dal file *Data.csv* in modo da estrarre solo le righe contenenti l'indicatore richiesto per l'analisi, ovvero *MS.MIL.XPND.GD.ZS*, codice che indica la percentuale di risorse investite per la difesa rispetto a quelle prodotte durante un intero anno. Viene, quindi, creato un RDD, che possiede il nome dello stato come chiave e il resto del record come oggetto.
I dati provenienti da MongoDB subiscono una trasformazione analoga, facilitata dall'accesso tramite chiave ai campi del documento preso in considerazione durante l'iterazione.
Dopo aver effettuato il join, attraverso il metodo standard offerto dal framework, si ottiene una struttura dati che possiede lo stato come chiave e una tupla di stringhe, ovvero la coppia dei record dei due file in cui c'è stato un match, come oggetto.
Questa struttura viene raffinata attraverso un'altra iterazione di map/reduce in modo tale da ottenerne una più compatta e di facile utilizzo per lo svolgimento dell'analisi vera e propria.
Il risultato delle modifiche restituisce un RDD avente come chiave una tupla composta dal nome dello stato e dal record contenente le informazioni necessarie per l'analisi e come oggetto tutti i vari attacchi subiti da quella particolare nazione.
Di seguito viene riportato la pseudocodifica del metodo che si occupa di eseguire l'esame proposto.

```
mostAttackedCountries.map(country,numOfAttacks)
result = previousJoin.join(mostAttackedCountries)
result.map(numOfAttacks,{country,getAverageDefenseExpenditure()})
result.sortBykey(descendent)
```

Il metodo implementato fa ricorso ad una delle analisi basilari precedentemente descritte, ovvero quella che si occupa di stilare una classifica delle nazioni che hanno subito più attacchi.
La struttura contenente questa classifica viene utilizzata per un secondo join che avviene con il risultato della procedura di Join analizzata in precedenza, al fine di ottenere un RDD in cui sono disponibili il numero di attacchi ricevuti ed il record in cui sono memorizzate le informazioni riguardanti la spesa affrontata

nel corso degli anni per la difesa interna.
Il risultato viene raffinato attraverso una procedura di map che si occupa anche di interpellare il metodo atto a calcolare la media aritmetica degli investimenti fatti per la sicurezza del paese preso in esame durante l'iterazione.
Questa procedura è molto semplice e non fa altro che utilizzare un *matcher* che ha il compito di estrarre i vari campi dal record del file, trasformarli in formato numerico e calcolarne la media.
Nella sezione della profilazione verranno poi analizzate le difficoltà sorte durante il matching automatico dei campi data la natura eterogenea dei file che possono essere presenti in un Datalake reale.
Il risultato finale viene poi salvato nel Datalake, la cui locazione è stata passata come parametro al momento del *deploy* del Jar nel framework.

## 4.2 Attacks Energy

Questa analisi mette in correlazione il numero di attacchi subiti da un paese con la sua produzione di energia tramite petrolio, analisi interessante che ha dimostrato come i paesi più attaccati siano anche quelli che hanno una più alta percentuale di produzione di energia.
Il procedimento risolutivo implementato per questa analisi è quasi identico a quello descritto in dettaglio in precedenza e non viene quindi approfondito.
Una considerazione che si può evincere dall'implementazione di queste due analisi poliglotte è che si possa automatizzare il procedimento di manipolazione e di join dei dati se si hanno a disposizione delle informazioni, a priori, sul contenuto del file e sulla disposizione degli attributi di interesse.
Questo compito verrà quindi analizzato nella sezione successiva con la descrizione del modulo *Profiler* che ha questo come fine ultimo.

# 5 Profiler

In questa sezione vengono illustrate le tecniche implementate per effettuare una profilazione dei dati, in questo progetto in particolare ci siamo posti l'obiettivo di identificare la colonna principale dei dataset del Data Lake, consistenti nelle entità rappresentanti gli Stati mondiali, per effettuare il join con il dataset su Mongo. Queste tecniche sono frutto di un iniziale studio sulla profilazione di dati e probabilmente non generalizzabili a qualsiasi tipo di dataset, soprattutto variando il dataset iniziale, ma sufficienti per il dominio del progetto.
Come infatti accennato, l'eterogeneità delle varie sorgenti è molto alta:

- un dataset può utilizzare per un'informazione mancante un valore *null*, oppure potrebbe utilizzare il numero 0, o ancora una stringa di convenzione come *"null"* o *"emptyField"*;
- un dataset può riportare una determinata entità utilizzando il nome effettivo, mentre un altro potrebbe utilizzare un sinonimo o un titolo attribuito a quella particolare entità. Ciò farebbe risultare i due *data case* non joinabili;
- non c'è uno standard fisso per il carattere delimitatore dei csv, infatti dalle due diverse fonti abbiamo estratto file con caratteri separatori differenti (”,” e ”;”), che hanno quindi richiesto *parsing* leggermente diversi;

## 5.1 Estrazione dei metadati

Il modulo di profilazione necessita in input il nome del dataset e il carattere separatore del csv (in base alla sorgente del file) e offre dei metodi per generare una struttura di metadati consistente in una mappa $< String, Long >$ che riporta per le possibili colonne principali il numero di valori diversi presenti al suo interno. La struttura *DatasetMetadata*, implementata come oggetto Java, presenta anche un campo intero, che riporta l'indice della colonna più probabilmente principale e su cui verrà effettuato il join. L'identificazione attualmente prevede semplicemente la selezione dell'indice che riporta il valore minore tra quelli presenti nella mappa, per mancanza di euristiche più robuste date le ristrette tempistiche per l'ideazione e le casistiche di basso numero analizzate.
Il Profiler filtra inizialmente le colonne partendo dall'assunzione che la presenza di diversi valori nulli o uguali a 0 sia caratterizzante di una colonna non di join, ma che riporta una qualche informazione quantitativa, che può avere valore pari a 0 o mancare del tutto.
Questa assunzione è stata dedotta dallo studio della mappatura delle colonne effettuata per popolare la struttura DatasetMetadata e ha dimostrato buone prestazioni nel filtrare colonne di valori numerici, ma anche colonne di valori booleani, quindi con uno basso numero di occorrenze diverse, chiaramente non indicate per un'eventuale join.
A questo punto è possibile invocare una funzione per salvare la struttura dati

ottenuta su MongoDB, nella apposita *collection "metadata"*, per utilizzarla al momento del join.

## 5.2 Join parametrico

Una volta salvati i metadati, essi possono essere richiamati in un modulo apposito, *ParametricJoin*.
In questa classe sono presenti metodi per importare il dataset da Mongo e quello del Data Lake passato come parametro di input ed i relativi metadati con una connessione Java Mongo nativa, data la dimensione contenuta.
Dai metadati viene quindi estratto l'indice della colonna ritenuta principale e avviato il join che segue l'implementazione di quello per l'analisi poliglotta.
Questo modulo prende in input il nome del dataset del Data Lake e il carattere separatore usato, che va ad influire sul sistema di *split* delle righe durante il join.

# 6 Risultati

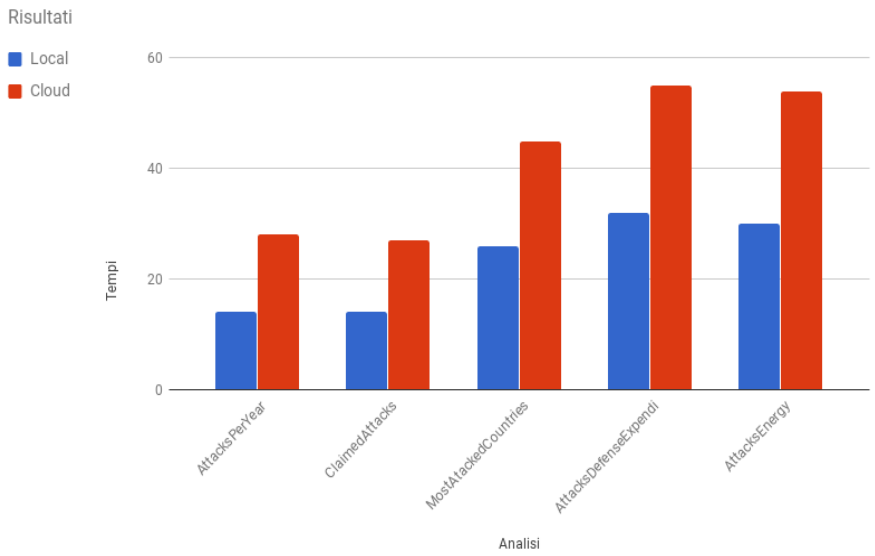
In questa sezione vengono presentati i risultati raccolti durante varie esecuzioni, eseguite in locale e su *Amazon Web Services*, delle analisi sviluppate.

## 6.1 Tempi delle analisi

| Analisi | Locale (s) | Cluster AWS (s) |
|---|---|---|
| MostAttackedCountries | 26 | 45 |
| AttacksPerYear | 14 | 28 |
| ClaimedAttacks | 14 | 27 |
| AttacksDefenseExpenditure | 32 | 55 |
| AttacksEnergy | 30 | 54 |

Table 1: Tempi delle esecuzioni in locale e cluster

La tabella sopra riportata indica la stima dei tempi ottenuti durante la messa in esecuzione delle analisi implementate.
Si nota come i tempi relativi al cluster siano maggiori rispetto alla versione locale, fattore dovuto principalmente ai più alti tempi di *startup* del cluster e alla dimensione esigua dei dataset utilizzati per gli esami.
Dei dataset più ingenti sicuramente appianerebbero le differenze temporali tra i due ambienti di esecuzione, sfruttando completamente la potenza di calcolo offerta da un cluster.
Non è stato possibile effettuare la prova sperimentale di questa affermazione a causa della difficoltà di trovare, sulle fonti a nostra disposizione, dataset di grandi dimensioni e correlabili per una analisi poliglotta come quella precedentemente presentata.
La differenza nelle prestazioni viene riportata in maniera visivamente più chiara nel grafico seguente.



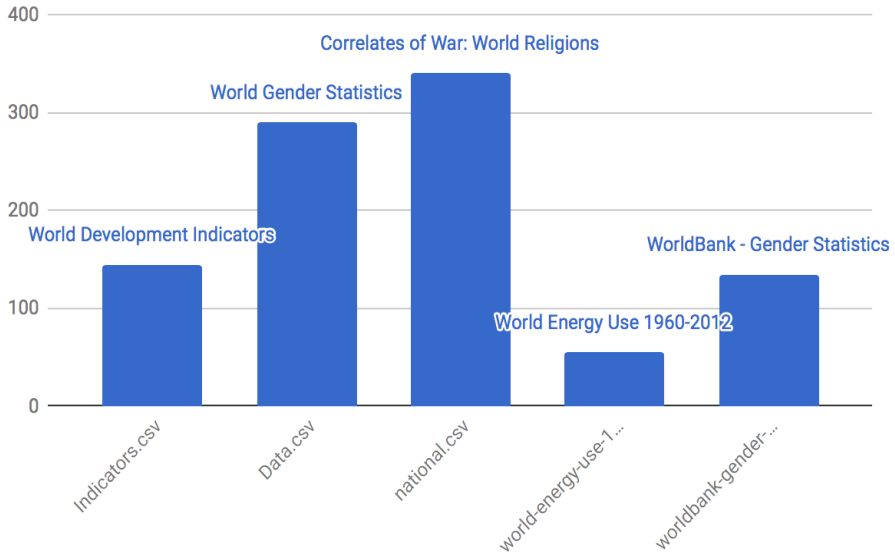

## 6.2 Tempi di profilazione

Nel seguito viene mostrato un report sulle informazioni e le tempistiche di analisi e profilazione dei vari dataset in input al Data Lake.

| Nome Dataset | Dimensione (MB) | Numero di colonne |
|---|---|---|
| World Development Indicators | 574,3 | 6 |
| World Gender Statistics | 41,7 | 60 |
| Correlates of War: World Religions | 0,6 | 79 |
| World Energy Use 1960-2012 | 0,383 | 6 |
| WorldBank - Gender Statistics | 741 | 6 |

Table 2: Descrizione dei dataset del Data Lake

I dataset provengono da due fonti differenti, Kaggle e Kapsarc, che utilizzano due formati differenti: il primo usa come delimitatore il carattere ",", mentre il secondo utilizza il ";" Si può notare la differenza di dimensione e numero di colonne tra i vari *csv*, fattori che andranno a influenzare i tempi di esecuzione.




Come risulta dall'andamento dei tempi per la profilazione, l'algoritmo è dipendente dal numero di colonne che compongono i *datasets*. Questo non sarebbe comunque un problema, poichè nel sistema finale esso verrebbe eseguito in background su ogni nuovo file in input al Data Lake, non creando quindi problemi di overhead per l'utilizzo dei dati già presenti, ma un'ottimizzazione del procedimento sarebbe comunque di beneficio.

# 7 Conclusioni e Sviluppi Futuri

La persistenza poliglotta rappresenta una problematica fondamentale da affrontare per il mondo dei Big Data del futuro.
Come ampiamente dimostrato nel corso di questa relazione, l'implementazione di tale sistema è possibile ma non priva di sfide.
La piattaforma sviluppata raggiunge l'obiettivo di favorire l'interoperabilità tra due basi di dati completamente differenti tra loro e di rendere possibili, di conseguenza, analisi incrociate tra questi due database.
Il modulo *Profiler*, inoltre, rende possibile una automatizzazione di questo processo ed è quindi fondamentale per una futura diffusione di sistemi poliglotti.
Possibili sviluppi della piattaforma sono infatti legati alla raffinazione del modulo di profilazione, con l'introduzione di funzionalità di interrogazione di *Knowledge Graph* in modo da contenere gli effetti della grande eterogeneità dei dati che possono rendere complicato il join tra sistemi diversi.
Tramite l'interrogazione di questi sorgenti di dati, infatti, è possibile relazionare dei dati semanticamente uguali ma sintatticamente definiti in maniera diversa.
Un esempio che conferma la precedente affermazione fa riferimento all'utilizzo di sinomini per indicare una stessa nazione, come *USA* o *United States of America* per gli Stati Uniti d'America.
L'interrogazione ad un grafo di conoscenza come *DBPedia* renderebbe possibile il join tra righe contenenti quelle due entità, limitando di conseguenza la perdita di informazione.