POLITECNICO DI TORINO
Facoltà di Ingegneria dell'Informazione
Corso di Laurea in Ingegneria Informatica

Tesi di Laurea

Progettazione e sviluppo di un'architettura Big Data per dati geospaziali

Relatori
Prof.ssa Elena Maria Baralis
Prof. Paolo Garza

Candidato
Roberto Arcomano

Marzo 2017

Indice generale

Indice generale 1
Indice delle figure 3
Capitolo 1 Introduzione 6
1.1 Descrizione problema affrontato 6
1.2 Sommario 6
Capitolo 2 Stato dell'arte della tecnologia Big Data 8
2.1 Architettura classica 8
2.2 Cosa sono i Big Data? 8
2.3 Infrastruttura Big Data 9
2.4 Requisiti Architettura Big Data 10
2.5 Vantaggi Big Data 10
2.6 Strumenti e Framework Big Data - Hadoop 11
2.6.1 HDFS 11
2.6.2 MAP-REDUCE 12
2.6.3 SPARK 15
2.6.4 Strumenti per gestire dati GeoReferenziali 16
2.6.5 Tecnologie di Cloud Computing 19
Capitolo 3 Architettura Big Data 20
3.1 Diagramma generico e analisi preliminare 20
3.2 Diagramma della soluzione applicata e analisi dettagliata 21
3.3 Struttura dati RDBMS 23
3.4 Migrazione da RDBMS a Blob Storage 24
3.4.1 Compatibilità file .CSV con librerie GeoSpark 24
3.4.2 Alternativa al Data Factory: Apache Sqoop 24
Capitolo 4 Analisi di dati geospaziali 26
4.1.1 Riepilogo delle analisi che verranno effettuate 26
4.1.2 Preparazione dell'ambiente 26
4.1.3 Inizializzazione 26
4.1.4 Filtro per un campo di descrizione 27
4.1.5 Controllo inclusione rettangolo singolo 28
4.1.6 Suddivisione di un rettangolo in più rettangoli 29
4.1.7 Matrice JavaPairRDD con occorrenze per rettangolo 30
4.1.8 JoinQuery 34
4.1.9 ChoroplethMap 34
4.1.10 HeatMap 36
4.1.11 Main finale 38
Capitolo 5 Analisi dettagliata architettura Big Data 39
5.1 Esportazione Dati RDBMS → HDFS 39
5.1.1 Microsoft Azure 39
5.1.2 Amazon EMR 41
5.2 Strumenti e fonti dati GeoReferenziali 46
5.2.1 Dati geografici - NaturalEarthData 46
5.2.2 Dati geografici – Geofabrik 48
5.2.3 Dati Eventi Sismici mondiali 49
5.2.4 Applicativi per la manipolazione dei dati geografici 51
5.2.5 Elaborazione dati georeferenziali 53

5.2.6 Note sull'esecuzione in Cluster 59
Capitolo 6 Analisi Costi 60
6.1 Costi Amazon AWS 60
6.1.1 Esempio di utilizzo servizi AWS 60
6.2 Costi Microsoft Azure 61
Capitolo 7 Note Conclusive 63
Capitolo 8 Dettagli su Microsoft Azure 64
8.1 Utilizzo del Data Factory 64
8.2 Creazione Cluster HDInsight 70
Capitolo 9 Tecnologie alternative di Cloud Computing 72
9.1 Creazione Cluster Ex novo su Amazon EMR 73
9.2 Clonazione di un Cluster preesistente su EMR 75
Capitolo 10 Bibliografia e Links 77
10.1 LIBRI 77
10.1.1 Java 77
10.1.2 Hadoop 77
10.1.3 Spark 77
10.2 APPLICATIVI SOFTWARE 77
10.2.1 Apache Hadoop 77
10.2.2 Apache Spark 77
10.2.3 Apache Sqoop 77
10.2.4 FreeGis 78
10.2.5 Qgis 78
10.3 LIBRERIE SOFTWARE 78
10.3.1 GeoSpark & Babylon 78
10.4 CLOUD PROVIDERS 78
10.4.1 Microsoft Azure 78
10.4.2 Amazon Web Services (AWS) 78
10.5 DATI GEOREFENZIATI 78
10.5.1 NaturalEarthData 78
10.5.2 OpenStreetMap Data Extracts 78
10.5.3 Database Eventi Sismici 79

Indice delle figure

Figura 2.1 Architettura a singolo Nodo 8
Figura 2.2 Architettura Multi Nodo 9
Figura 2.3 Struttura HDFS 11
Figura 2.4 Map Reduce 12
Figura 2.5 Esempio di Envelope formato da 2 punti 17
Figura 3.1 Diagramma Generico Architettura Big Data 20
Figura 3.2 Diagramma della soluzione applicata 22
Figura 3.3 Campi tabella Reports 23

Figura 4.1 Esempio di Controllo Inclusione Punti in un rettangolo 28
Figura 4.2 Suddivisione Rettangolo in rettangoli più piccoli 29
Figura 4.3 Matrice Occorrenze Punti per Rettangolo 31
Figura 4.4 JavaRDD<String> vs JavaPairRDD <String,Long> 32
Figura 4.5 JavaRDD<Envelope> vs JavaPairRDD < Envelope,Long> 33
Figura 4.6 Esempio ChoroplethMap 36
Figura 4.7 Esempio HeatMap 37
Figura 5.1 Elenco Cluster EMR 41
Figura 5.2 Impostazioni di Sicurezza Cluster EMR.....41
Figura 5.3 Security Group EMR 42
Figura 5.4 Firewall Rules per EMR 43
Figura 5.5 Security Group per RDS 44
Figura 5.6 Sito Web Natural Earth Data 47
Figura 5.7 Sito Web GeoFabrik 48
Figura 5.8 Esempio tramite QGis di dati presi da GeoFabrik.....49
Figura 5.9 Sito Web Eventi Sismici mondiali.....49
Figura 5.10 Esempio esportazione dati da Sito Web www.emsc-csem.org.....50
Figura 5.11 Logo QGIS 51
Figura 5.12 Aggiunta Vettore su QGIS 51
Figura 5.13 Selezione zona da esportare su QGIS 52
Figura 5.14 Selezione oggetti interni alla zona selezionata 52
Figura 5.15 Esportazione vettore QGIS 53
Figura 5.16 Immagine finale prodotto alla fine dell'elaborazione di tutti gli oggetti georeferenziati 58
Figura 6.1 Esempio Fatturazione Servizi Amazon AWS.....61
Figura 6.2 Esempio fatturazione servizi Microsoft Azure 62
Figura 8.1 Creazione Data Factory 64
Figura 8.2 Menu Data Factory 65
Figura 8.3 Creazione Pipeline 65
Figura 8.4 Selezione Data Store Sorgente.....66
Figura 8.5 Selezione Connessione Database.....66
Figura 8.6 Impostazione Query per l'esportazione.....67
Figura 8.7 Selezione DataStore di Destinazione.....67
Figura 8.8 Selezione Blob Storage.....68
Figura 8.9 Selezione Directory HDFS di Destinazione 68
Figura 8.10 Dettagli per l'Esportazione CSV 69
Figura 8.11 Dettagli per l'ottimizzazione dell'esportazione 69
Figura 8.12 Riepilogo Esportazione.....70
Figura 8.13 Creazione Cluster HDInsight.....70
Figura 8.14 Selezione Contenitore Blob Storage 71
Figura 8.15 Dettagli sul Cluster 71
Figura 9.1 Elenco Cluster EMR 73
Figura 9.2 Creazione Cluster EMR ex-novo.....73

Figura 9.3 Scelta del tipo di Cluster 74
Figura 9.4 Clonazione Cluster 75
Figura 9.5 Esportazione comando di creazione Cluster 76

Capitolo 1 Introduzione

1.1 Descrizione problema affrontato

Il problema affrontato nella presente Tesi riguarda la necessità di trovare un'architettura che permetta la gestione di una grandissima quantità di dati GeoReferenziati proveniente da un Database Relazionale per analisi successive: per grandissima quantità di dati qui s'intende: occupazione file dati > 1PB.

L'architettura generale all'interno della quale si inserisce il problema affrontato prevede di fatto 2 componenti: una prima componente "ON-LINE" di tipo RDMBS e una seconda componente di tipo "OFF-LINE", tipicamente con tecnologia Big Data che, partendo dai dati generati dalla prima componente, sia in grado di effettuare delle elaborazioni su una quantità di dati con ordine di grandezza maggiore della prima componente e che permetta di effettuare la stessa tipologia di Check sopra descritti.

La componente ON-LINE è associata ad un'architettura RDBMS, mentre per componente di tipo OFF-LINE è stata scelta un'architettura di tipo Big Data.

In particolare la componente Big-Data può essere attivata solo quando richiesta, quindi in modalità ON-DEMAND, permettendo di sfruttare le risorse di elaborazione solo quando è strettamente necessario.

Nella presente Tesi verrà discussa l'architettura, focalizzandosi in particolare sulla seconda componente di tipo Big Data, sull'interazione con la prima componente e sulle librerie utilizzate per la gestione dei dati GeoReferenziati.

1.2 Sommario

Per iniziare si passerà in rassegna allo Stato dell'Arte della tecnologia Big Data, partendo dai presupposti teorici, dalla necessità di questa nuova tecnologia e dalle sue principali caratteristiche.

Successivamente si descriveranno i principali Framework di tipo Big Data presenti sul mercato, le relative estensioni per gestire i dati GeoReferenziali e le tecnologie di supporto necessarie per l'utilizzo dei servizi Big Data (Cloud).

Sarà quindi effettuata un'analisi approfondita dell'architettura necessaria per poter gestire i dati in oggetto, in maniera teorica con l'ausilio di schemi architetturali, e in maniera più concreta, con la descrizione delle soluzioni tecnologiche scelte; verrà poi effettuato un

confronto con altre tecnologie disponibili e verrà prodotto un risultato finale facilmente comprensibile.

Verrà infine fornita una serie di dati sperimentali come risultato delle elaborazioni effettuate, in particolare:

1. Tempi di migrazione dati dalla componente ON-LINE a quella OFF-LINE
2. Tempi di elaborazione richiesti, sulla componente OFF-LINE, per la generazione dell'output di tipo "mappe di calore"
3. Dettaglio dei costi necessario in base all'architettura scelta.

La Tesi finirà con le conclusioni su quello che è stato fatto e cosa è possibile fare, partendo dal lavoro svolto fin qui, per poterlo integrare e/o migliorare.

Capitolo 2 Stato dell'arte della tecnologia Big Data

2.1 Architettura classica

L'architettura di elaborazione classica prevede fondamentalmente 3 componenti hardware:

1. CPU
2. RAM
3. DISK

In generale, per effettuare delle elaborazioni si segue la seguente sequenza:

1. Vengono caricati dal DISK alla RAM una porzione dei dati da elaborare
2. La CPU elabora i dati
3. I risultati vengono salvati su DISK.

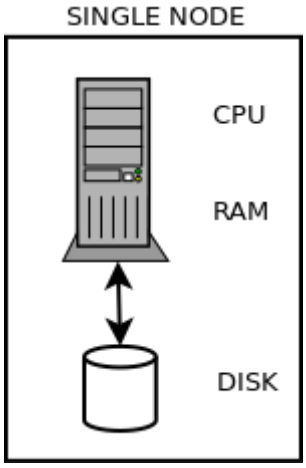


Figura 2.1 Architettura a singolo Nodo

Quando si ha a che fare con grandi moli di dati (si parla dell'ordine di grandezza di PB o decine di PB. 1PB=10^{15}Bytes), l'infrastruttura classica non permette di ottenere dei risultati validi in tempi soddisfacenti.

2.2 Cosa sono i Big Data?

La definizione più universalmente accettata che si può trovare di Big Data è la seguente:

"Una raccolta di dati così estesa, in termini di capacità occupata, eterogeneità e complessità da considerare le tecnologie tradizionali obsolete e richiedere l'utilizzo di nuove architetture, tecniche, algoritmi e strumenti analitici più adatti".

Analizzando la definizione bisogna capire cosa si intende esattamente per capacità occupata, eterogeneità e complessità:

1. Capacità occupata: spazio misurato in Bytes dalla struttura che ospita i nostri dati: ha senso parlare di Big Data quando la capacità occupata è dell'ordine del PB (10^{15} Bytes) o delle decine di PB
2. Eterogeneità (Variety): tipologia di dati che non segue un formato predefinito ma che può variare (es.: dati testuali, numerici, immagini, audio, video, date, GPS point, ecc.)
3. Complessità: oltre a cambiare la tipologia dei dati, può cambiare la qualità dei dati in esame (Veracity) e la rapidità di generazione dei dati può essere anche molto elevata (Velocity).

Quando la capacità dei dati su cui si vogliono eseguire delle elaborazioni diventa troppo grande, non è più sufficiente utilizzare un'infrastruttura semplice, ad esempio basata su un RDBMS, ma è necessario migrare ad una nuova infrastruttura.

2.3 Infrastruttura Big Data

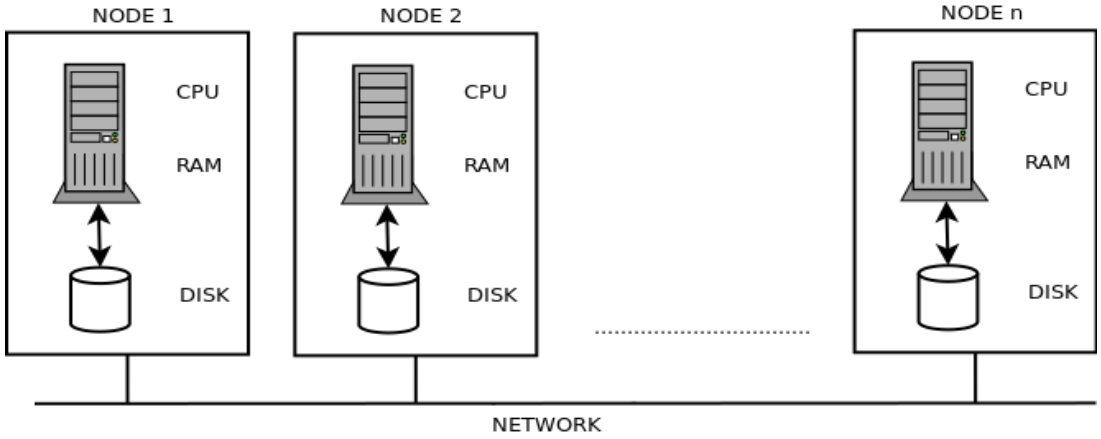


Figura 2.2 Architettura Multi Nodo

La soluzione proposta dall'architettura Big Data prevede l'impiego di molteplici nodi, connessi attraverso la rete, come nello figura seguente.
Viene utilizzato un protocollo distribuito che si occupa di suddividere l'intero volume di dati da elaborare in n parti (una per ogni nodo); così facendo, si garantisce che ogni nodo possa avere accesso in locale alla porzione dati di propria competenza (in realtà si preferisce avere almeno 2 nodi che abbiano la stessa informazione per motivi di ridondanza).

A questo punto il software suddividerà l'elaborazione di un singolo job in n istanze, una per ciascun nodo e operante solo su dati presenti in locale sul nodo stesso (funzionalità di MAP) e farà partire gli n job sugli n nodi.

Al termine dell'elaborazione il software effettuerà una raccolta di tutti i risultati e produrrà un unico output (funzionalità di REDUCE).

Questa è l'idea di base con cui funziona l'algoritmo di MAP-REDUCE all'interno di una infrastruttura Big Data.

2.4 Requisiti Architettura Big Data

Per poter utilizzare un'architettura Big Data come quella sopra descritta bisogna soddisfare una serie di requisiti:

1. Utilizzare N elaboratori, ognuno con una prestabilita caratteristica hardware (di solito vengono utilizzati dei Commodity Servers con 32GB di RAM, con più processori e disco locale con prestazioni adeguate)
2. Connessione di rete ad alta velocità, tipicamente 1Gbps
3. Utilizzo di 2 livelli di Switch: nodes Switch e backbone Switch (con prestazioni a 10Gbps)
4. Software distribuito per poter delocalizzare i dati, suddividere il carico tra i nodi e raccogliere i risultati.

2.5 Vantaggi Big Data

Il vantaggio più rilevante della utilizzo della tecnologia Big Data consta di 2 elementi:

1. Fattibilità realizzativa di una soluzione ad un problema riguardante grandi quantità di dati
2. Ridotti costi dell'architettura, grazie all'utilizzo di diversi nodi uguali tra loro che suddividono il carico

NB.: E' importante notare che, il 2. punto può essere notevolmente ridotto grazie alla tecnologia di Cloud Computing, come si vedrà più avanti.

2.6 Strumenti e Framework Big Data - Hadoop

Esistono numerose implementazioni dell'architettura Big Data: qui ci concentreremo essenzialmente sulla soluzione Open Source di Apache Hadoop Spark.

2.6.1 HDFS

All'interno del framework Hadoop, la componente che si occupa di delocalizzare il volume di dati suddividendolo tra tutti i nodi presenti è HDFS.

HDFS è un vero e proprio File System Distribuito che si occupa di tenere sincronizzati i nodi: un file presente sul NAME NODE viene suddiviso in "CHUNK", cioè in porzioni di file e suddiviso tra i nodi in modo da mantenerne almeno una duplice copia (come detto prima per motivi di ridondanza).

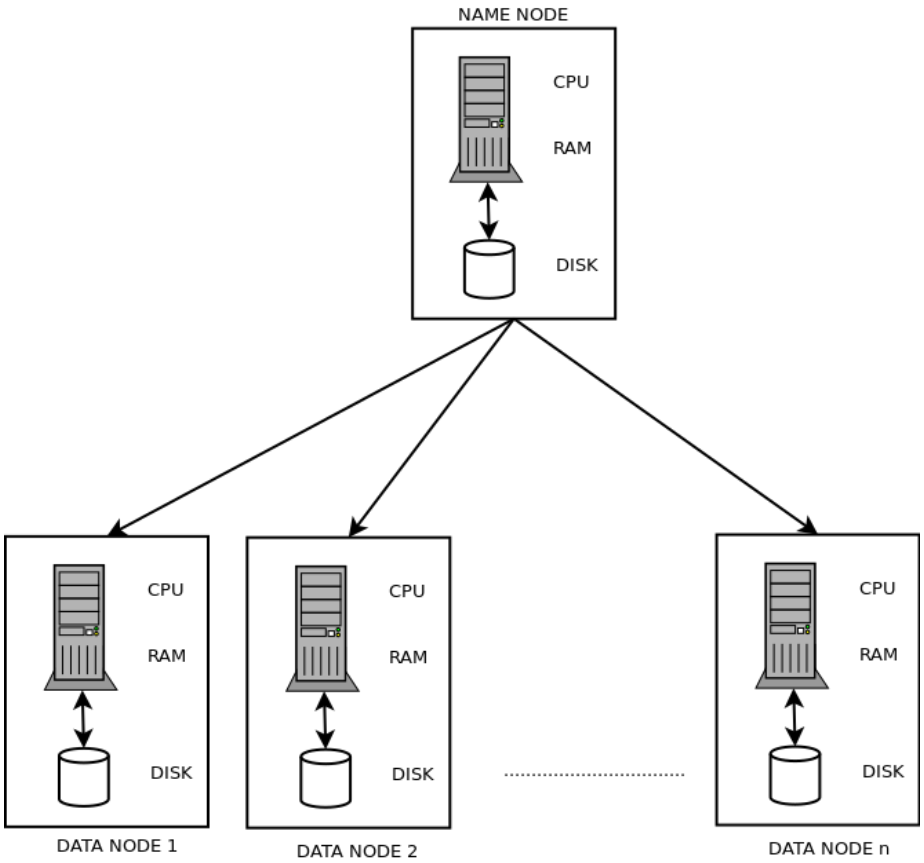


Figura 2.3 Struttura HDFS

Un grande vantaggio di HDFS è quello di essere “Scalabile”, cioè di consentire senza problemi l’aggiunta di un nuovo nodo senza perturbare il funzionamento dell’architettura nel suo complesso

2.6.2 MAP-REDUCE

Come accennato in precedenza, il primo approccio su Hadoop alla programmazione distribuita è stato il paradigma “Map-Reduce” che si compone sostanzialmente di 2 parti:

1. MAPPER: funzione utilizzata per lavorare sulla singola porzione di dato locale (CHUNK) sull’i-esimo nodo
2. REDUCER: funzione che raccoglie i dati dai nodi che hanno eseguito il MAPPER e crea il risultato finale.

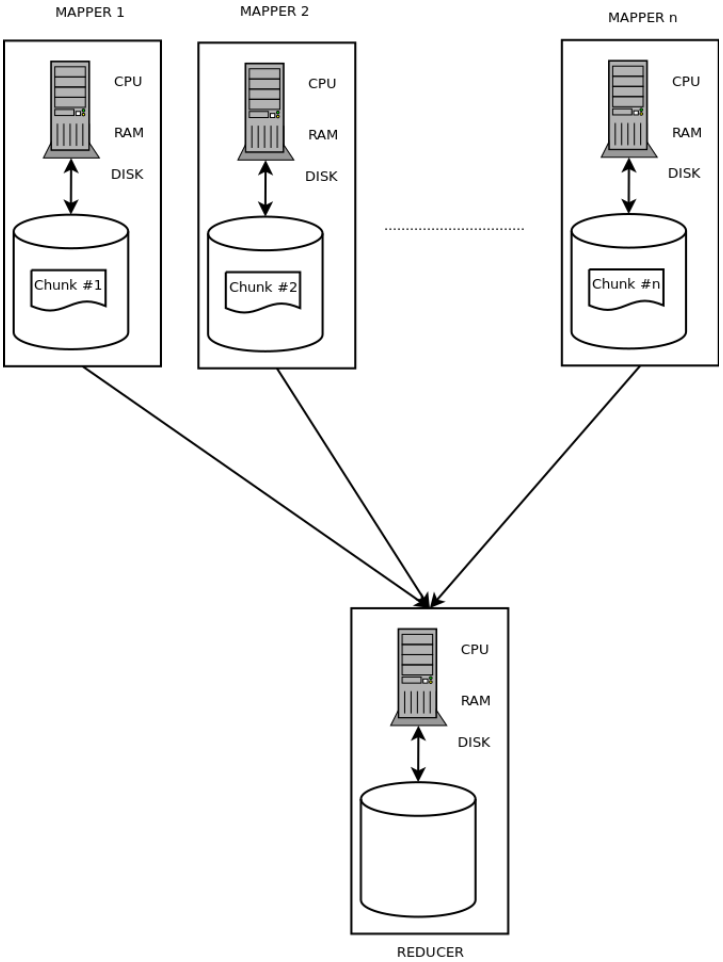


Figura 2.4 Map Reduce

Driver

```
package it.polito.bigdata.hadoop.example;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;


/**
 * Programma MapReduce
 */
public class DriverBigData extends Configured implements Tool {

  @Override
  public int run(String[] args) throws Exception {

    Path inputPath;
    Path outputDir;
    int numberOfReducers;
            int exitCode;


            // Lettura parametri
    numberOfReducers = Integer.parseInt(args[0]);
    inputPath = new Path(args[1]);
    outputDir = new Path(args[2]);


    Configuration conf = this.getConf();

    // Crea JOB
    Job job = Job.getInstance(conf);

    // Assegna nome al JOB
    job.setJobName("Esempio");


    // Imposta PATH INPUT
    FileInputFormat.addInputPath(job, inputPath);

    // Imposta PATH INPUT
    FileOutputFormat.setOutputPath(job, outputDir);


    // Imposta classe del Driver
    job.setJarByClass(DriverBigData.class);


    // Imposta formato input
    job.setInputFormatClass(TextInputFormat.class);

    // Imposta formato output
    job.setOutputFormatClass(TextOutputFormat.class);


    // Imposta classe MAPPER
    job.setMapperClass(MapperBigData.class);
```

```
    // Imposta le classi per Key e Value
    job.setMapOutputKeyClass(Text.class);
    job.setMapOutputValueClass(IntWritable.class);


    // Imposta la classe REDUCER
    job.setReducerClass(ReducerBigData.class);


    // Imposta le classi per Key e Value in output
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);

    // Imposta numero dei REDUCER
    job.setNumReduceTasks(numberOfReducers);


    // Esegue il job e rimane in attesa per il suo termine
    if (job.waitForCompletion(true)==true)
                exitCode=0;
    else
                exitCode=1;


    return exitCode;
  }

  /** Main
   */


  public static void main(String args[]) throws Exception {
                // Usa la classe ToolRunner per configurare e lanciare l'applicazione Hadoop
    int res = ToolRunner.run(new Configuration(), new DriverBigData(), args);

    System.exit(res);
  }
}
```

Mapper

```
package it.polito.bigdata.hadoop.example;

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

/**
 * Mapper
 */
class MapperBigData extends Mapper<
                LongWritable, // Tipo Key di Input
                Text,         // Tipo Value di Input
                Text,         // Tipo Key di Output
                IntWritable> {// Tipo Value di Output


    protected void map(
            LongWritable key,
            Text value,
            Context context) throws IOException, InterruptedException {

            // Suddivide le frasi in parole
```

```
        String[] words = value.toString().split("\\s+");

        // Per ogni parola
        for(String word : words) {
            // Calcola il minuscolo
            String cleanedWord = word.toLowerCase();

            // Scrive l'accoppiata (word, 1)
            context.write(new Text(cleanedWord), new IntWritable(1));
        }
    }
}
```

Reducer

```
package it.polito.bigdata.hadoop.example;

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

/**
 * Reducer
 */
class ReducerBigData extends Reducer<
                        Text,          // Tipo Key di Input
                        IntWritable,   // Tipo Value di Input
                        Text,          // Tipo Key di Output
                        IntWritable> {  // Tipo Value di Output


    @Override
    protected void reduce(
        Text key,
        Iterable<IntWritable> values,
        Context context) throws IOException, InterruptedException {

        int occurrances = 0;

        // Per ogni parola uguale => somma
        for (IntWritable value : values) {
            occurrances = occurrances + value.get();
        }
        // Scrive il totale delle occorrenze
        context.write(key, new IntWritable(occurrances));
    }
}
```

2.6.3 SPARK

Apache Spark rappresenta un'ottimizzazione del paradigma MAP-REDUCE.

Le principali migliorie sono:

1. Utilizzo più efficiente della RAM per minimizzare gli accessi all'HDFS

2. Principio del RDDs, Resilient Distribuited Data Sets, un oggetto distribuito tra più nodi per rendere più agevole lo sviluppo delle applicazioni Hadoop e per ottimizzare le prestazioni.

Gli RDDs sono quindi delle "variabili distribuite" che hanno particolari caratteristiche:

1. Una volta create sono Invarianti, quindi possiamo considerarle come delle costanti
2. Su di esse si possono eseguire 2 tipi di operazioni:
   I. Transformations (map, filter, join,…) che restituiscono un nuovo RDD
   II. Actions (count, collect, save, …) che danno un risultato "locale"
3. Le Transformations sono operazioni "Lazily", cioè non vengono eseguite subito ma solo quando, su di esse viene eseguita una Action: questo per motivi di prestazioni.

```
package it.polito.bigdata.spark.esempio;

import org.apache.spark.api.java.*;
import org.apache.spark.api.java.function.Function;
import org.apache.spark.SparkConf;


public class SparkDriver {
  public static void main(String[] args) {
    String inputPath  = args[0]; // Primo parametro: Percorso input
    String outputPath = args[1]; // Secondo parametro: Percorso output


    // Crea un oggetto configuration e imposta il nome all'applicazione
    SparkConf conf = new SparkConf().setAppName("Esempio Spark");


    // Crea oggetto JavaSparkContext
    JavaSparkContext sc = new JavaSparkContext(conf);

    // Legge dal file fornito come parametro in input
    JavaRDD<String> RDD = sc.textFile(inputPath);

    JavaRDD<String> esempiRDD = RDD.filter(
      new Function<String, Boolean>() {
                public Boolean call(String logLine) {
                  // Filtra solo le righe che contengono la parola "esempi"
                  return logLine.toLowerCase().contains("esempi");
                }
              }
    );
    // Salva il risultato sul percorso fornito dal secondo parametro
    esempiRDD.saveAsTextFile(outputPath);
    // Chiusura dello Spark context
    sc.close();
  }
}
```

2.6.4 Strumenti per gestire dati GeoReferenziali

In questi anni, molte tecnologie di sviluppo si stanno adeguando per interfacciarsi con nuove strutture dati, in particolare con il formato GeoReferenziale.

Per gestire i suddetti dati, Apache Spark può essere esteso con le librerie GeoSpark, che prevedono le classi:

1. Point, che permette di gestire un punto singolo, memorizzando le sue coordinate geografiche in termini di Longitudine e Latitudine
2. Envelope, che permette di definire un rettangolo geografico, cioè un'area geografica delimitata da 2 coordinate: per ricavare gli altri 2 punti di quest'area è sufficiente tenere fissa una coordinata e variare l'altra, poi fare il viceversa.

Figura 2.5 Esempio di Envelope formato da 2 punti

3. Polygon, che rappresenta un'area delimitata da più punti.
4. MultiPolygon, un unione di più polygon

Mentre invece, per quanto riguarda gli oggetti di tipo RDD, l'estensione GeoSpark ne introduce di nuovi, definendoli SRDD (Spatial-RDD), per poter gestire meglio le operazioni tra strutture dati RDD.

I tipi di SRDD definiti sono:
1. PointRDD,
2. RectangleRDD,
3. PolygonRDD,
4. LineStringRDD,
5. CircleRDD

Come per gli RDD, gli SRDD definiscono una costante a livello di Cluster, i cui dati possono risiedere in qualunque nodo; in aggiunta vengono definiti dei metodi e degli attributi aggiuntivi, accessibile dalla classe, come ad esempio:
- public JavaRDD<Point> getRawPointRDD() che permette di passare da un SRDD al suo corrispondente RDD
- public void setRawPointRDD(JavaRDD<Point> rawPointRDD), l'opposto del precedente, che serve per passare da un RDD classico ad un PointRDD
- public boolean contains(Coordinate) che determina se la coordinata passata in ingresso è contenuta all'interno dell'oggetto chiamante
- public void expandToInclude(Coordinate), che espande l'oggetto attuale per includere la coordinata passata in ingresso
- public boolean intersects(Coordinate) che determina se l'oggetto attuale interseca la coordinata passata in ingresso.

Negli esempi sopra citati, in luogo della classe Coordinate è possibile specificare la classe Envelope.

Altri esempi notevoli sono quelli relativi agli insiemi, ad esempio:
- public Envelope intersection(Envelope), che restituisce l'intersezione tra l'oggetto attuale e quello passato
- public MultiPolygon union(Polygon), che restituisce l'unione tra l'oggetto attuale e quello passato, andando quindi a costruire un nuovo oggetto di MultiPolygon
- public Polygon difference(Polygon), che restituisce il risultato dell'operazione insiemistica di "differenza" tra 2 insiemi: quindi verrà restituito il Polygon originale dopo aver tolto il Polygon passato in ingresso.

2.6.5 Tecnologie di Cloud Computing

Per poter utilizzare in maniera produttiva un'architettura di tipo Big Data è necessario disporre di hardware generalmente costoso, che viene dedicato a tal fine: questo limite renderebbe molto difficile l'utilizzo di tecnologie Big Data.
Oggigiorno è facile disporre in maniera flessibile di nuovo hardware grazie alla tecnologia Cloud Computing: è sufficiente effettuare una rapida registrazione presso il portale web che il fornitore mette a disposizione e, usando i wizard previsti, creare / modificare nuove virtual machines o cluster (come nel caso di Big Data, dove è richiesta la creazione specifica di un Cluster Hadoop).

A seconda della tipologia di Cloud Computing, verrà addebitata una tariffa che può essere forfait o oraria.

Durante l'intervallo di disponibilità delle risorse, queste:

- saranno accessibili attraverso la rete Internet;
- potranno interagire quindi con altri servizi presenti su Internet;
- configurabili "ON DEMAND", a seconda del fornitore, permettendo quindi un ridimensionamento delle risorse sia in SCALE-UP che in SCALE-DOWN, in tempo reale

Tra i fornitori più conosciuti di Cloud Computing ci sono:

- Amazon Web Services (AWS)
- Microsoft Cloud Azure

Nel presente testo verrà analizzata l'architettura proposta da Microsoft.

Su Azure, per attivare un cluster Apache Hadoop con Spark è necessario istanziare una risorsa di tipo cluster HDInsight.

Capitolo 3 Architettura Big Data

3.1 Diagramma generico e analisi preliminare

Per analizzare il problema può essere utile uno schema che mostri l'architettura in oggetto con le principali componenti

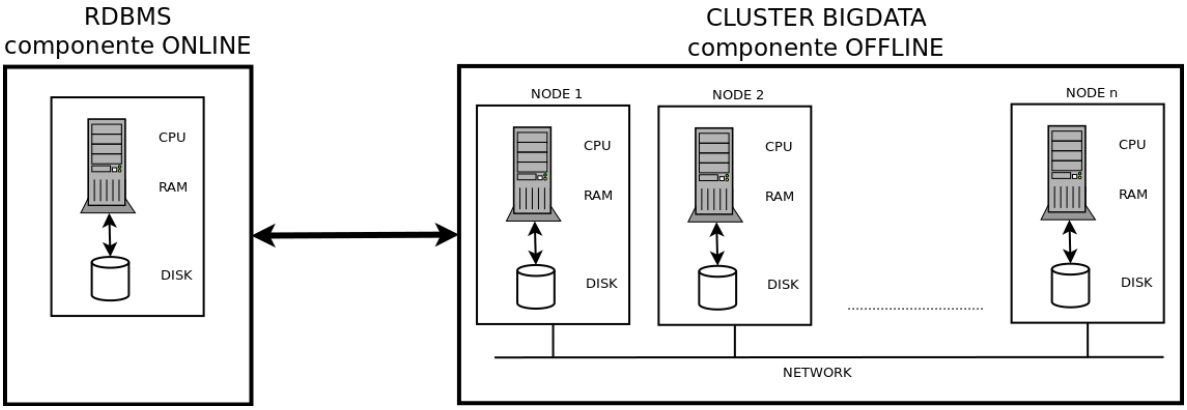


Figura 3.1 Diagramma Generico Architettura Big Data

La componente ONLINE si occupa di:

1. di ricevere le informazioni direttamente dal mondo esterno, tipicamente tramite delle DML di tipo Insert ed eventualmente delete per cancellare dati non più utilizzati.
2. Effettuare delle elaborazioni e restituire risultati, anche con l'ausilio di librerie georeferenziate di tipo DB-relazionale

La componente OFFLINE si occupa invece di:

1. implementare la funzionalità di "storico", periodicamente ricevendo i dati più vecchi dalla componente ONLINE prima della loro eliminazione e tenerli memorizzati in una struttura dati opportuna, tipicamente non di tipo relazionale.
2. Consentire delle analisi su grandi quantità di dati (in quanto lavorerebbe con un storico contenente molti più dati della componente ONLINE), tramite librerie georeferenziali.
3. Restituire il risultato dei calcoli effettuati una volta completati, in qualche forma (Web Services, email, ecc.)

Le 2 componenti ONLINE e OFFLINE, appartengono a 2 categorie informatiche molto diverse: quella ONLINE può essere definita anche "REALTIME" nel senso che tutte le

operazioni su di essa, sia inserimenti, cancellazioni che analisi ed interrogazioni, e anche la tipologia stessa dei dati che contiene, sono relativi all'istante in cui vengono eseguite, mentre la componente OFFLINE può essere definita di tipo "BATCH" perché, ogni operazione che la coinvolge, richiede un certo tempo per essere eseguita (sia l'importazione dei dati che le analisi) ed è quindi naturalmente associabile a concetti come JOB.

Un'altra differenza tra le 2 componenti viene attribuita dai fornitori che mettono a disposizione la loro architettura, il costo: esso risulterà tendenzialmente più basso per la componente ONLINE, semplicemente perché richiede poco hardware e risorse di elaborazione, mentre quella OFFLINE richiederà l'istanziazione di numerosi nodi (possono andare da 4 fino a diverse centinaia o migliaia).

Da quest'ultimo punto si evince che, la prima componente non porta a particolari criticità economiche e quindi può rimanere attiva e funzionante a tempo indeterminato.

La seconda componente, invece, ha in se una difficoltà economica intrinseca: è richiesta la persistenza dei dati a tempo indefinito, ma le risorse economiche per poterla tenere attiva e funzionante sono troppo elevate: si vedrà più avanti come affrontare questo problema.

Un'ultima considerazione preliminare: l'architettura mostrata qui non è soltanto valida nel caso di gestione e analisi di dati GeoReferenziati ma può essere utilizzata in molti ambienti dove è preesistente un RDBMS ormai utilizzato pressochè ovunque, dando la possibilità di effettuare delle nuove operazioni, magari con obiettivi strategici come ad esempio marketing: questo lavoro, di solito, viene già effettuato tramite software di Data Warehouse (DWH).

3.2 Diagramma della soluzione applicata e analisi dettagliata

Come accennato nel diagramma generico, è necessario trovare una soluzione al problema dell'elevato costo dell'infrastruttura Big Data: la tecnologia di Cloud Computing di Microsoft Azure risolve questo problema introducendo una nuova componente nell'architettura: il Blob Storage.

Il Blob Storage, come suggerisce il termine, è un "contenitore" generico dalle molteplici applicazioni ma essenzialmente è una porzione di volume logico a basso costo fruibile attraverso la piattaforma di Cloud Computing.

Il vantaggio principale del Blob Storage, nelle applicazioni Big Data, risiede nel poter essere utilizzato durante la creazione di un cluster HDInsight (nome dato alla soluzione Microsoft basata sul Cluster Hadoop) come se fosse un file system di tipo HDFS: in questo

modo, una volta creato il Cluster Hadoop, i singoli nodi potranno lavorare sull'HDFS in maniera trasparente.
Nota: L'utilizzo del Blob Storage all'interno di una infrastruttura Big Data apre uno scenario di discussione su quali siano le implicazioni da un punto di vista di performance dell'architettura: queste considerazioni verranno approfondite nel capitolo Analisi Prestazioni.

In questo modo, si potranno mantenere attive e funzionanti solo le 2 componenti a basso costo Sql Server e Blob Storage, mentre l'oneroso Cluster HDInsight potrà essere disattivato e distrutto per evitare costi inutili. Quando servirà di nuovo potrà essere creato ex-novo (previa attesa di diversi minuti per la creazione delle relative risorse), specificando come il Blob Storage come unità HDFS.

Nota: All'interno dell'architettura Microsoft Azure, per azzerare il costo di una componente non è sufficiente disattivarla temporaneamente ma è necessario distruggerla in maniera definitiva.

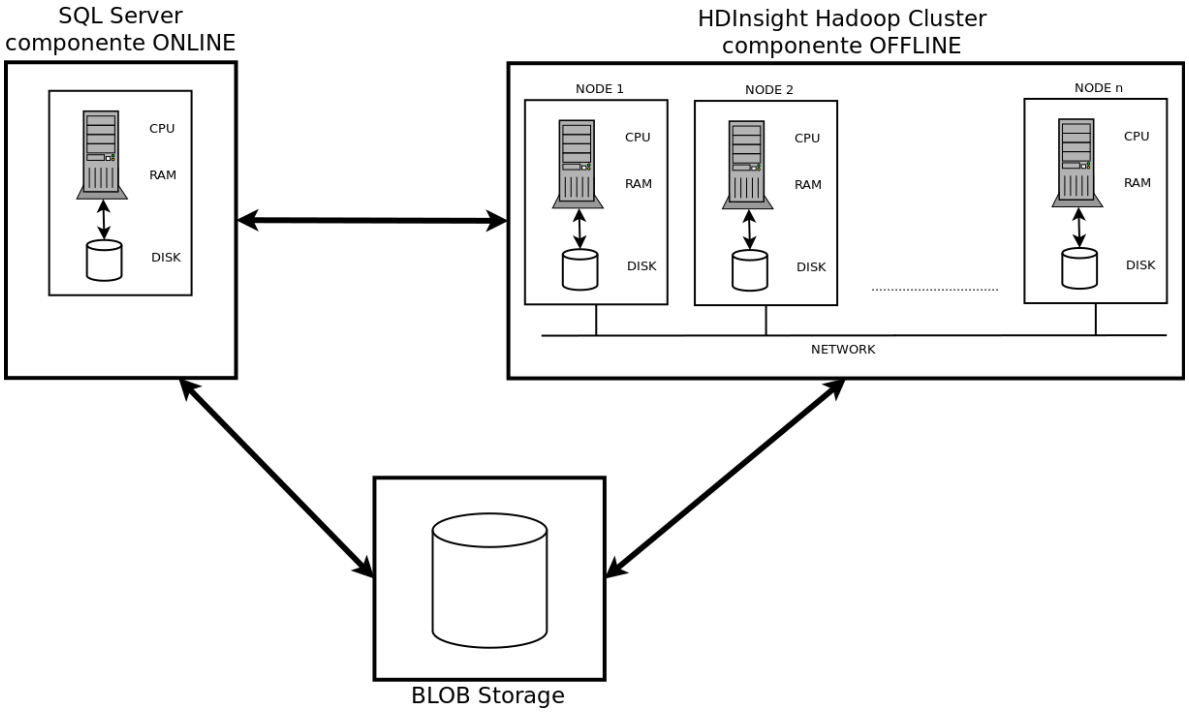


Figura 3.2 Diagramma della soluzione applicata

3.3 Struttura dati RDBMS

Per il problema affrontato, la struttura dati utilizzata prevede un'unica tabella, REPORTS con i campi del tipo descritto nella figura 3.3

Column Name		Data Type	Length	Not Null
id	1	int	10	☑
Description	2	varchar	250	☐
Picture	3	varchar	100	☐
Position	4	geography	2.147.483.647	☐
AugmentedPosition	5	varchar	1	☐
HPL	6	varchar	1	☐
CreateDate	7	datetime	23	☐
UpdateDate	8	datetime	23	☐
DeleteDate	9	datetime	23	☐
part_id	10	int	10	☐

Figura 3.3 Campi tabella Reports

Si può notare il campo "Position" di tipo geography, che rappresenta un coordinata contenente longitudine e latitudine.

Per la gestione dei dati georeferenziati in tabella è necessario fare un di attenzione: in particolare per effettuare gli inserimenti si utilizza la seguente sintassi:
geography::Point(@longitude, @latitude, 4326)

dove:

- @longitude è un numero decimale che indica la longitudine
- @latitude è un numero decimale che indica la latitudine
- 4326 è una costante utilizzata per utilizzare lo standard mondiale geodetico del 1984
- geography::Point è un metodo che genera il valore codificato per poter essere inserito in tabella.

Nota: il carattere "@" viene utilizzato negli script Sql Server per fare riferimento a delle variabili, ad esempio "@longitude" da riferimento al valore della variabile chiamata longitude.

3.4 Migrazione da RDBMS a Blob Storage

La prima attività da analizzare è la migrazione dei dati dal database Relazionale all'ambiente Hadoop; in particolare il database in oggetto su Azure è Microsoft Sql Server, mentre la destinazione scelta, per i ragionamenti fatti in precedenza, è il Blob Storage.
La prima soluzione per la migrazione da RDBMS a Blob Storage consiste nell'utilizzare il prodotto di Cloud Azure "Data Factory": per maggiori informazioni si veda l'Appendice A.

3.4.1 Compatibilità file .CSV con librerie GeoSpark

La procedura di esportazione appena vista produce un file con formato .CSV usando il carattere "," come separatore tra i campi.
Le librerie GeoSpark importano i dati considerando solo campi di tipo GeoReferenziale, in sostanza 2 numeri separati da virgole, il primo che rappresenta la Longitudine e il secondo la Latitudine.

Per rendere compatibile, quindi, i dati esportati dal Data Factory di Azure con le librerie GeoSpark ci sono 2 possibilità:

1. cambiare la query vista sopra esportando solo i dati riguardanti la posizione ed eliminando i caratteri "POINT (" e ")" creati durante l'esportazione delle coordinate
2. prima di eseguire i job di analisi di GeoSpark, eseguire un job preventivo per la riformattazione dei dati.

La soluzione più efficiente è sicuramente la 1, in quanto non richiede un'ulteriore elaborazione su Spark preliminare alle analisi successive.

3.4.2 Alternativa al Data Factory: Apache Sqoop

Invece di utilizzare il Data Factory, si può fare uso di un tool speciale sviluppato direttamente da Apache che mette in comunicazione un database relazione con il file system HDFS necessario per le elaborazioni BigData.

Il tool si chiama Sqoop, viene lanciato su un ambiente Linux che possa avere accesso, da un lato all'ambiente RDBMS (nel caso in oggetto SQL Server), dall'altro al file system HDFS per poter scrivere il risultato dell'esportazione dei dati delle tabelle.

Un esempio di esecuzione del comando sqoop è il seguente

```
sqoop import --connect "jdbc:mysql://localhost/ireact?user=root&password=cloudera" --table report
```

Il comando specificato si connetterà al database relazionale (nel caso del comando il tipo di database è mysql, ma con le librerie Java corrette si potrà accedere a qualunque database

relazionale, Oracle, SQL Server, ecc.), accederà alla tabella report ed esporterà tutti le colonne di tutti i record inserendoli in un file testo.

E' possibile specificare una serie di parametri a Sqoop per poter selezionare righe / colonne della tabella: ad esempio tramite il parametro "where" si può indicare una condizione di where, oppure con il parametro "query" si può addirittura esplicitare la query da eseguire.

Come ogni altro comando eseguito su Hadoop, anche Sqoop viene eseguito in parallelo tramite tasks separati: a tal fine è previsto il parametro "num-mappers" che permette di specificare il numero dei mapper da utilizzare durante l'esportazione, portando quindi a creare diverse connessioni separate all'RDBMS da parte dei nodi del cluster Hadoop.

Con Sqoop è possibile anche eseguire l'operazione opposta: partendo da un file di testo (di solito in formato .CSV) eseguire l'importazione sul database relazionale.

In questo caso il comando da eseguire è il seguente:

$ sqoop export --connect jdbc:mysql://db.example.com/foo --table bar --export-dir /results/bar_data

Nota riepilogativa: con "sqoop import" vengono presi i dati dall'RDBMS e inseriti in un file di testo su HDFS, mentre con "sqoop export" vengono utilizzati i dati di un file presente su HDFS ed eseguite le insert su RDBMS.
Purtroppo la convenzione utilizzata è l'inversa, nel senso che il termine "esportazione" usato quando si porta di RDBMS corrisponde ad una "importazione" quando si parla di Sqoop e viceversa, in quanto quest'ultimo usa il punto di vista di HDFS.

Capitolo 4 Analisi di dati geospaziali

Una volta esportati i dati dal Database relazionale e una volta pronto il Cluster Hadoop, è possibile partire con le analisi di tipo Big Data utilizzando le librerie georeferenziali GeoSpark disponibili.

I dati messi a disposizione dall'RDBMS sostanzialmente rappresentano delle coordinate singole, quindi del punti geografici.

4.1.1 Riepilogo delle analisi che verranno effettuate

1. Filtrare x descrizione
2. Controllo inclusione su Envelope costante: contare inclusioni
3. Suddivisione rettangolo in griglia di NxM rettangoli
4. Calcolo occorrenze punti per ogni rettangolo
5. JoinQuery
6. HeatMap

4.1.2 Preparazione dell'ambiente

Per poter utilizzare le librerie GeoSpark è necessario effettuare dei controlli sull'ambiente Hadoop:

1. Controllare la versione di Spark installata con "spark-submit --version"
2. Scaricare la versione corretta di GeoSpark in base alla versione di Spark installata (ci sono 2 possibilità, versione di Spark 1.x e Spark 2.x).
   URL GeoSpark: https://github.com/DataSystemsLab/GeoSpark/releases
3. Preparare script di lancio di Spark che richiami la libreria corretta, tramite il parametro "--jars"
4. In testa allo script, la prima istruzione da eseguire deve essere la cancellazione della directory di output che verrà utilizzata da Spark
5. effettuare i primi test in locale (parametro –deploy-mode client) e una volta che tutto funziona spostarsi su HDFS (parametro –deploy-mode cluster)

4.1.3 Inizializzazione

```
package it.polito.bigdata.spark.AnalisiGeoSpark;

import org.apache.spark.SparkConf;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.api.java.JavaSparkContext;
```

```
import org.apache.spark.api.java.function.Function;
import org.datasyslab.geospark.spatialRDD.PointRDD;

import com.vividsolutions.jts.geom.Coordinate;
import com.vividsolutions.jts.geom.Envelope;
import com.vividsolutions.jts.geom.GeometryFactory;
import com.vividsolutions.jts.geom.Point;

public class SparkDriver {
	public static void main(String[] args) {
		SparkConf conf = new SparkConf().setAppName("Analisi GeoSpark");
		JavaSparkContext sc = new JavaSparkContext(conf);

		// Input file
		String location = "in/reports";

		// Legge ogni riga del file
		JavaRDD <String> rdd = sc.textFile(location);
	}
}
```

Vengono definiti degli import che serviranno per le chiamate successive.
La classe principale effettua delle inizializzazioni preliminari (SparkConf e JavaSparkContext), viene definito il percorso del file input e si carica su un RDD di tipo "String" tutto il file.

4.1.4 Filtro per un campo di descrizione

```
	private static JavaRDD <String> filtraPerDescrizione(JavaRDD<String> rdd) {

		// Filtro in base al primo campo
		JavaRDD <String> rddFiltered = rdd.filter(
				new Function<String,Boolean>() {
					public Boolean call(String row) {
						return row.split(",")[0].contains("3");
					}
				}
		);
		return rddFiltered;
	}
```

Il metodo si occupa, passato un RDD di tipo String, di restituire solo le righe che hanno come primo campo una stringa che contiene il carattere "3".

4.1.5 Controllo inclusione rettangolo singolo

Un primo esempio ed importante utilizzo di dati georeferenziati è il controllo di inclusione di un insieme di punti all'interno di un determinato rettangolo (o Envelope).

Per effettuare tale controllo, le librerie GeoSpark mettono a disposizione il metodo "contains(Coordinate)" all'interno della classe Envelope.

Il metodo filtraPerRettangolo si occuperà di filtrare i punti in base alla loro appartenenza al rettangolo passato in input.

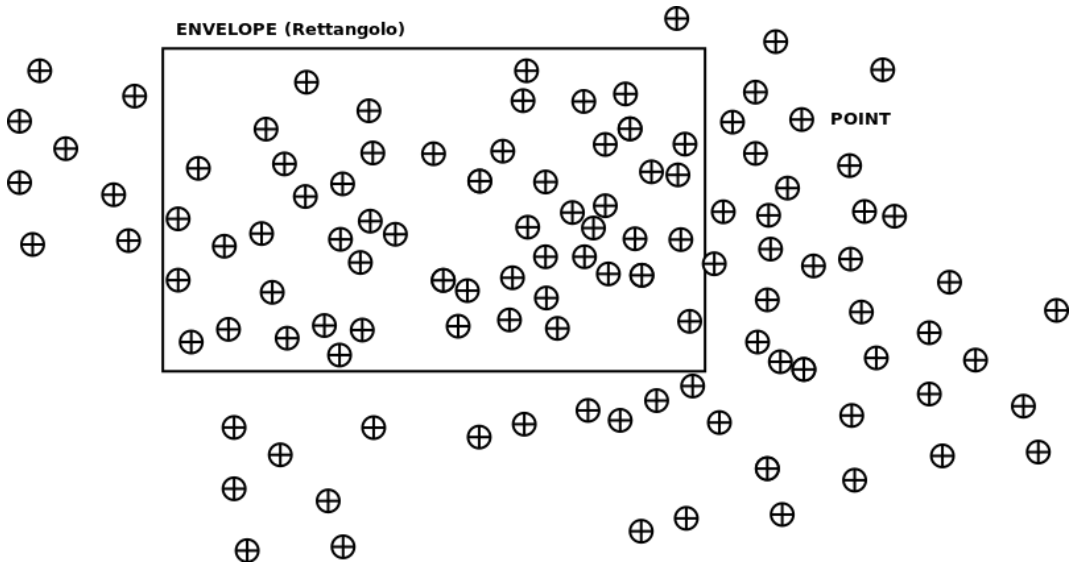


Figura 4.1 Esempio di Controllo Inclusione Punti in un rettangolo

```
private static JavaRDD <Point> filtraPerRettangolo(JavaRDD<Point> javaRDDPoint, final Envelope envelope) {
        JavaRDD <Point> underEnvelope = javaRDDPoint.filter(
                new Function <Point,Boolean>() {
                        public Boolean call(Point point) {
                                if (envelope.contains(point.getX(),point.getY()))
                                        return true;
                                return false;
                        }
        });
        return underEnvelope;
    }
```

4.1.6 Suddivisione di un rettangolo in più rettangoli

Per poter effettuare dei test interessanti, si è deciso di creare un metodo che, dati in ingresso:

- un rettangolo
- fattore N – divisione orizzontale
- fattore M – divisione verticale

generi in output un array di rettangoli più piccoli dell'originale che, rispettando i valori N ed M, possono essere pensati per coprire tutti insieme la stessa area occupata dal rettangolo originale.

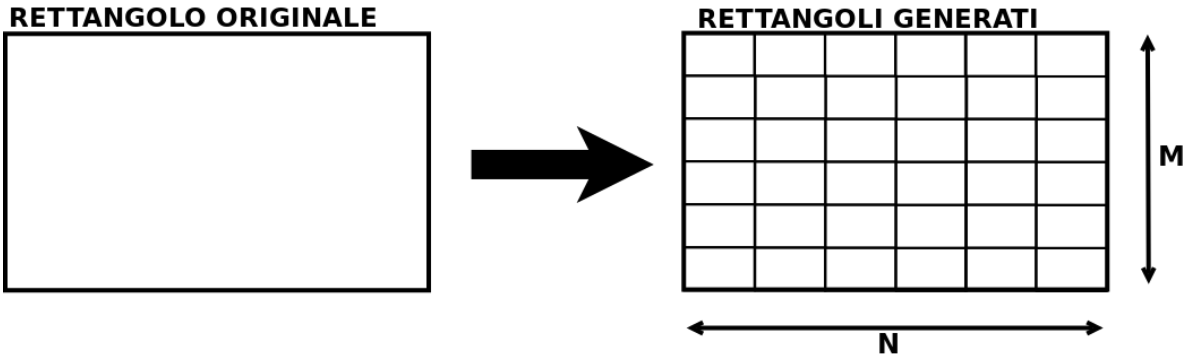


Figura 4.2 Suddivisione Rettangolo in rettangoli più piccoli

Il metodo può essere realizzato come segue:

- Calcolo dei valori di deltaX e deltaY, partendo dal rettangolo originale e dai 2 parametri in ingresso, chiamati numX (N) e numY (M)
- Calcolo della coordinata in alto a sinistra del rettangolo originale per stabilire il punto di partenza per il calcolo delle coordinate dei rettangolini: punto (x0,y0)
- Creazione primo rettangolino che ha:
  - coordinata in alto a sinistra uguale a quella del punto di partenza in alto a sinistra del rettangolo originale (x0,y0)
  - coordinata in basso a destra = (x0 + deltaX, y0 + deltaY)
  - man mano che ci si sposta verso il basso, cambieranno solo le coordinate Y, incrementando sia il punto iniziale (in alto a sinistra) sia il punto finale (in basso a destra) di 1 deltaY
  - terminata la 1º colonna, si ritorna alla 1º riga della 2º colonna, e così via.

Segue il codice del metodo:

private static RectangleRDD calcolaRettangoli(JavaSparkContext sc, Envelope rettangolo, int numX, int numY) {

```
        List <Envelope> rettangoli = new ArrayList<Envelope>();

        // Calcola deltaX e deltaY
        double deltaX = (rettangolo.getMaxX()-rettangolo.getMinX())/numX;
        double deltaY = (rettangolo.getMaxY()-rettangolo.getMinY())/numY;

        // Recupera prima coordinata del rettangolo iniziale (in alto a sinistra)
        double x0 = rettangolo.getMinX();
        double y0 = rettangolo.getMinY();

        for (int x = 0; x < numX; x++) {
                for (int y = 0; y < numY; y++) {
                        double x1,x2,y1,y2;
                        x1 = x0 + deltaX*x;
                        y1 = y0 + deltaY*y;
                        x2 = x0 + deltaX*(x+1);
                        y2 = y0 + deltaY*(y+1);
                        Envelope rettangolino = new Envelope(x1,x2,y1,y2);
                        rettangoli.add(rettangolino);
                }
        }
        JavaRDD<Envelope> rettangoliRDD = sc.parallelize(rettangoli);
        return new RectangleRDD(rettangoliRDD);
}
```

Il metodo appena descritto è fondamentale per poter creare una griglia di “zone” georeferenziate a partire da un’unica zona.
7

4.1.7 Matrice JavaPairRDD con occorrenze per rettangolo

Il precedente calcolo di inclusione dei punti interni ad un rettangolo può essere esteso prendendo in considerazione un array di rettangoli cui corrispondono le relative occorrenze dei punti inclusi, in ogni rettangolo.

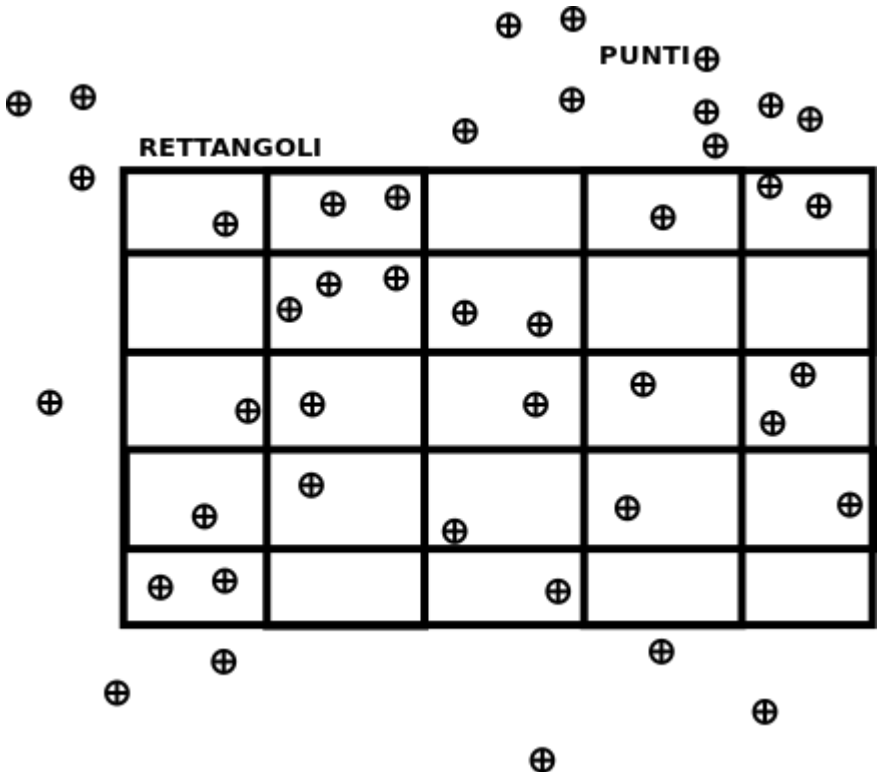


Figura 4.3 Matrice Occorrenze Punti per Rettangolo

Per poter effettuare questo tipo di calcolo, il framework di Spark mette a disposizione un particolare tipo di struttura dati, il JavaPairRDD.

Mentre la struttura dati JavaRDD rappresenta sostanzialmente un array di elementi singoli, il JavaPairRDD è un array che contiene accoppiate chiave/valore; in molti casi il JavaPairRDD può risultare utile, ad esempio quando si vuole mettere in relazione 2 tipologie di dati differenti oppure effettuare dei conteggi per ogni elemento.

L'accoppiata chiave/valore viene identificata con il termine "Tupla".

Nella figura viene rappresentata la differenza tra la struttura JavaRDD e JavaPairRDD, utilizzando un tipo di dato semplice "String" per JavaRDD e "<String,Long>" per il JavaPairRDD, ma possono essere utilizzate tutte le classi valide previste su Java.

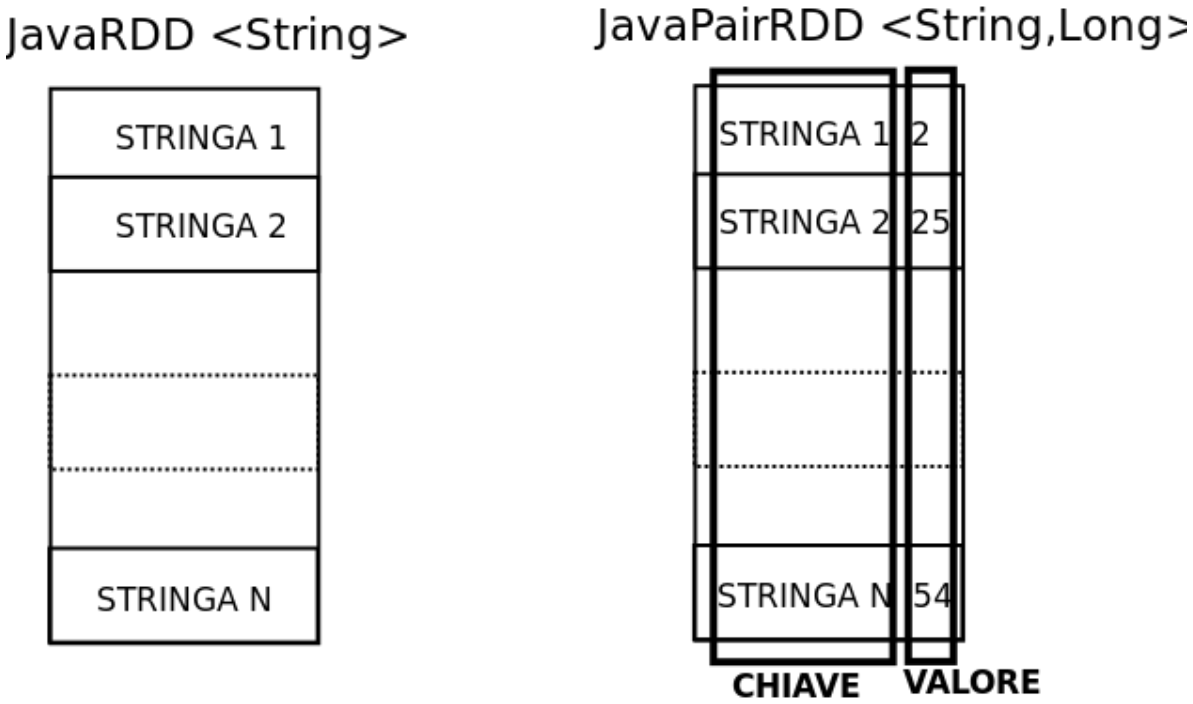


Figura 4.4 JavaRDD<String> vs JavaPairRDD <String,Long>

Per quanto riguarda il caso specifico, la figura 4.5 mostra le strutture dati da prendere in oggetto.

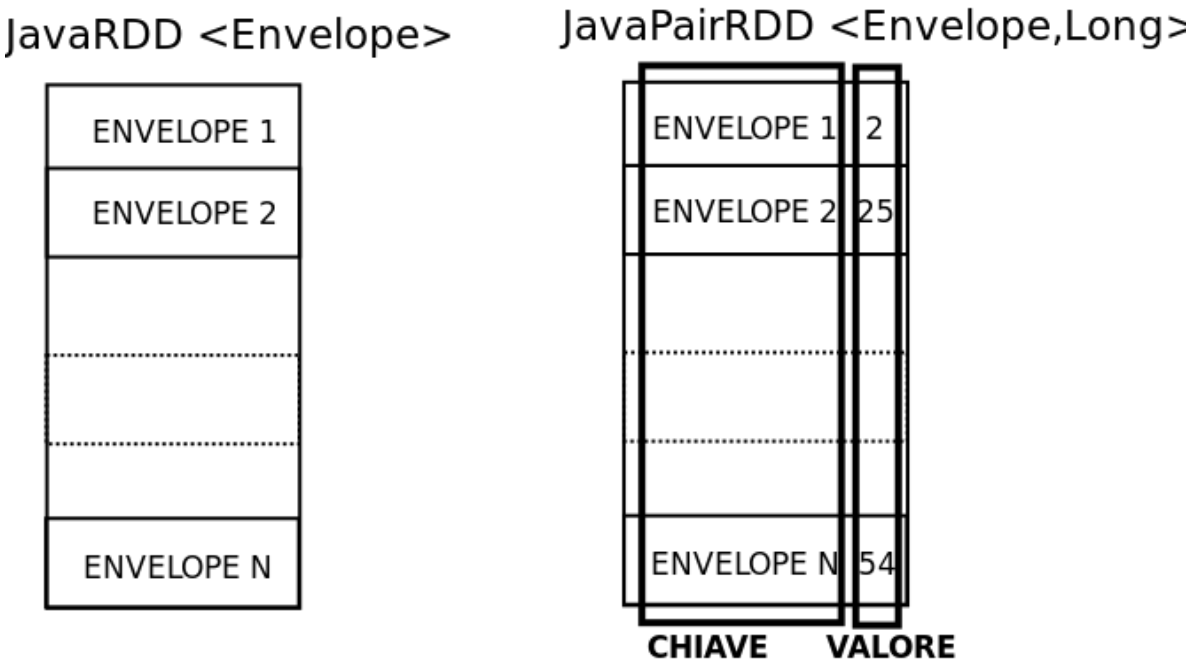


Figura 4.5 JavaRDD<Envelope> vs JavaPairRDD < Envelope,Long>

Il codice del metodo che costruisce la struttura dati di tipo JavaPairRDD è il seguente:

```
private static JavaPairRDD<Envelope,Long> numeroPuntiPerRettangolo(final JavaRDD<Point> points, JavaRDD<Envelope> envelopes) {
        JavaPairRDD<Envelope,Long> occorrenzePerRettangolo = envelopes.mapToPair(new PairFunction<Envelope,Envelope,Long>() {
                public Tuple2<Envelope,Long> call(Envelope envelope) {
                        Long contaPointInEnvelope = new Long(filtraPerRettangolo(points,envelope).count());
                        return new Tuple2<Envelope,Long>(envelope,contaPointInEnvelope);
                }
        });
        return occorrenzePerRettangolo;
}
```

Viene utilizzato il metodo mapToPair per passare dalla struttura JavaRDD alla JavaPairRDD; all'interno del metodo call viene richiamato il metodo visto in precedenza, filtraPerRettangolo che restituiva JavaRDD<Point>: in questo caso ci serve solo il numero di elementi, quindi richiamiamo la count() e ritorniamo così l'accoppiata Rettangolo/Numero punti contenuti.

4.1.8 JoinQuery

Il framework di GeoSpark mette a disposizione la classe JoinQuery per poter effettuare i calcoli di raggruppamento e conteggio punti georeferenziali in relazione ad ogni Envelope. Più in generale è possibile effettuare lo stesso tipo di calcolo prendendo in considerazione la struttura Polygon che permette di definire, come dice il nome, un generico poligono con N lati, quindi un'area geografica delimitata.
I 2 metodi in oggetto sono:

public static JavaPairRDD<Envelope, Long> SpatialJoinQueryCountByKey(SpatialRDD spatialRDD,RectangleRDD queryRDD,boolean useIndex)

che considera strutture dati rettangolari e

public static JavaPairRDD<Polygon, Long> SpatialJoinQueryCountByKey(SpatialRDD spatialRDD,PolygonRDD queryRDD, boolean useIndex)

che utilizza poligoni.

Questi metodo possono sfruttare degli indici spaziali per migliorare le prestazioni nei calcoli che effettuano: senza entrare nei dettagli, gli indici in questione sono gli R-Tree e i Quad-Tree.

4.1.9 ChoroplethMap

La classe ChoroplethMap fa uso dei concetti visti in precedenza, cioè permette di visualizzare il risultato della joinQuery creando N rettangoli con, ognuno, contenente un'informazione a livello di numero di punti presenti. Il metodo per visualizzare il numero di punti è una scala di colore variabile la cui intensità dipende dal numero di punti presenti in quel dato rettangolo.

Segue il codice che utilizza la classe ChoroplethMap:

```
private static void choroplethMap(JavaSparkContext sc, SpatialRDD oggetti1,RectangleRDD oggetti2, String
nomeFile, Envelope area) throws Exception {
	oggetti1.spatialPartitioning(GridType.RTREE);
	oggetti2.spatialPartitioning(oggetti1.grids);
	oggetti1.buildIndex(IndexType.RTREE,true);
	JavaPairRDD<Envelope,Long>		joinResult		=
JoinQuery.SpatialJoinQueryCountByKey(oggetti1,oggetti2,true);

	ChoroplethMap visualizationOperator = new ChoroplethMap(1000,600,area,false);
	visualizationOperator.CustomizeColor(255, 255, 255, 255, Color.RED, true);
	visualizationOperator.Visualize(sc, joinResult);
```

```
        ScatterPlot frontImage = new ScatterPlot(1000,600,area,false);
        frontImage.CustomizeColor(0, 0, 0, 255, Color.GREEN, true);
        frontImage.Visualize(sc, oggetti2);

        OverlayOperator overlayOperator = new OverlayOperator(visualizationOperator.pixelImage);
        overlayOperator.JoinImage(frontImage.pixelImage);

        NativeJavaImageGenerator imageGenerator = new NativeJavaImageGenerator();
        imageGenerator.SaveAsFile(overlayOperator.backImage,                "./target/nuovo/"+nomeFile,
ImageType.PNG);
}
```

Il punto chiave del codice è il richiamo del metodo SpatialJoinQueryCountByKey che di fatto relazione i rettangoli con il numero dei punti.

Il parametro area viene utilizzato per marcare i limiti dell'immagine, a livello di coordinate georeferenziali: nell'esempio mostrato sotto l'area è stata creata tramite l'istruzione seguente:

Envelope area = new Envelope(-126.790180,-64.630926,24.863836,50.000);

Se si guardano le coordinate specificate, si vedrà che l'intervallo specificato è un rettangolo che include tutto il territorio geografico degli Stati Uniti.

In figura 4.6 è presente un esempio di immagine risultante dall'elaborazione.

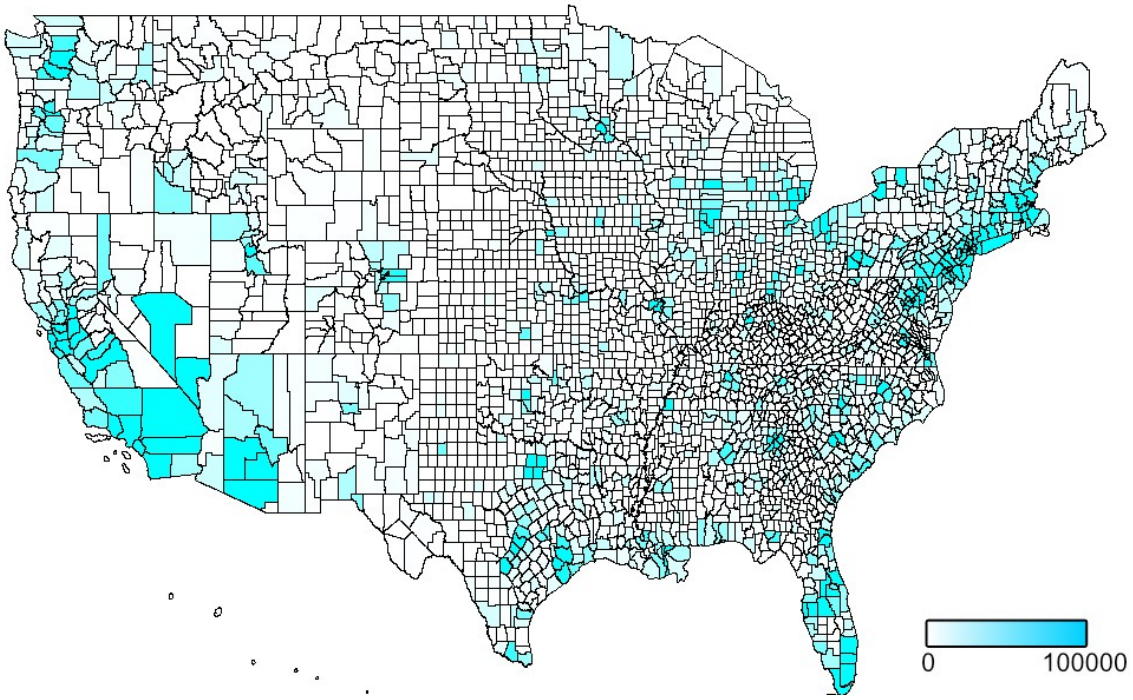


Figura 4.6 Esempio ChoroplethMap

4.1.10 HeatMap

La classe Heatmap permette di rappresentare punti, rettangoli e poligoni: il criterio utilizzato per la visualizzazione si basa sul concetto di sottolineatura del dato, permettendo di cambiare il colore di una certa zona in base a quanti oggetti vengono a sovrapporsi su quella zona.
Questa modalità ricorda un po' le mappe di calore, che tendono ad andare verso il rosso in caso di temperatura elevata da un estremo e ad andare verso il blu in caso di temperatura bassa, dall'altro estremo.

Codice di esempio:

```
public static void heatMap(JavaSparkContext sc, SpatialRDD oggetti,String nomeFile, Envelope area) throws Exception {
        HeatMap visualizationOperator = new HeatMap(800,500,area,false,3);
        visualizationOperator.Visualize(sc, oggetti);
        NativeJavaImageGenerator imageGenerator = new  NativeJavaImageGenerator();
        imageGenerator.SaveAsFile(visualizationOperator.pixelImage,     "./target/heatmap/"+nomeFile, ImageType.PNG);
```

}

La figura 4.7 mostra un esempio di immagine prodotta di HeatMap

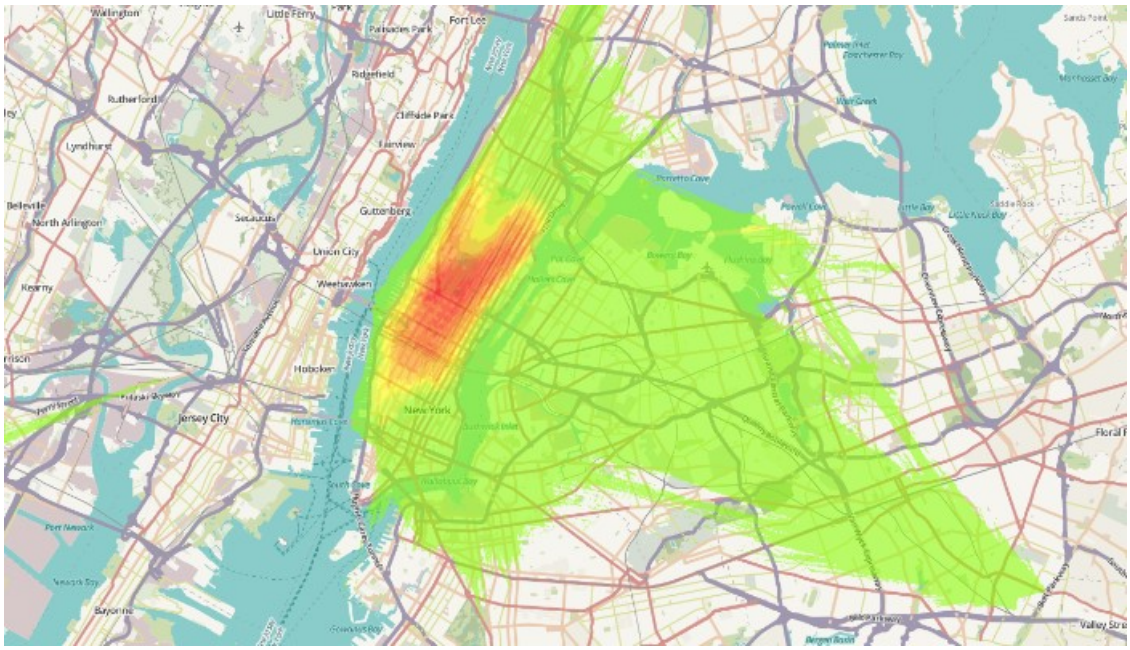


Figura 4.7 Esempio HeatMap

4.1.11 Main finale

Qui sotto è riportato il codice del main utilizzato per effettuare i test su GeoSpark:

```
public static void main(String[] args) {
	SparkConf conf = new SparkConf().setAppName("Analisi GeoSpark");
	JavaSparkContext sc = new JavaSparkContext(conf);

	// Input file
	String point = "in/arealm.csv";
	String polygon = "in/primaryroads-polygon1.csv";
	String outputLocation = "./output";

	// Area Stati Uniti
	Envelope area = new Envelope(-126.790180,-64.630926,24.863836,50.000);

	PointRDD punti = new PointRDD(sc, point, 0, FileDataSplitter.CSV, true, 1);
	PolygonRDD poligoni = new PolygonRDD(sc, polygon,  FileDataSplitter.CSV, false, 10);
	RectangleRDD rettangoli = calcolaRettangoli(sc, area, 100, 100);

	try {
		heatMap(sc,punti,"punti",area);
		heatMap(sc,rettangoli,"rettangoli",area);
		heatMap(sc,poligoni,"poligoni",area);

		scatterPlot(sc,punti,"punti",area);
		scatterPlot(sc,rettangoli,"rettangoli",area);
		scatterPlot(sc,poligoni,"poligoni",area);
		choroplethMap(sc,punti, rettangoli,"punti_e_rettangoli",area);
	} catch(Exception e) {
		;
	};

	sc.close();
}
```

Capitolo 5 Analisi dettagliata architettura Big Data

Verranno qui riportate una serie di risultati su test effettuati, relativamente a:

1. Esportazione dati dalla componente ONLINE (RDBMS) verso la componente OFFLINE (HDFS)
2. Elaborazione dati, tutta interna all'architettura Big Data di Hadoop Spark su dati presenti in HDFS.

5.1 Esportazione Dati RDBMS → HDFS

5.1.1 Microsoft Azure

La prima prova che verrà presa in considerazione sarà relativa all'esportazione da SQL Server utilizzando il DataFactory di Microsoft: la tabella su Sql Server contiene 2 milioni di records (contenente ognuno 1 punto georeferenziale).

La prova prevede l'esportazione completa della tabella, senza considerare alcun filtro. Il file prodotto è di circa 700MB in formato CSV e il tempo impiegato per l'esportazione è di circa 16 minuti.

Come alternativa al Data Factory è stato anche utilizzato, nelle prove, l'applicativo Sqoop: il risultato è stato, per l'esportazione di 2 milioni di records, di 4 minuti. Segue il comando sqoop eseguito e l'output generato

Comando sqoop:
sqoop import --connect "jdbc:sqlserver://server-sql-1.database.windows.net;database=reactdb" --username adminsql --password <password> --driver com.microsoft.sqlserver.jdbc.SQLServerDriver --table "dbo.reports" --target-dir output --m 1

output:

```
roby@hn0-ireact:~/sqoop$ date > data;.sqoop import --connect "jdbc:sqlserver://server-sql-1.database.windows.net;database=reactdb" --username adminsql --password <password> --driver com.microsoft.sqlserver.jdbc.SQLServerDriver --table "dbo.reports" --target-dir output --m 1 ;date>>data
Warning: /usr/hdp/2.5.4.0-121/accumulo does not exist! Accumulo imports will fail.
Please set $ACCUMULO_HOME to the root of your Accumulo installation.
17/03/04 17:25:07 INFO sqoop.Sqoop: Running Sqoop version: 1.4.6.2.5.4.0-121
17/03/04 17:25:07 WARN tool.BaseSqoopTool: Setting your password on the command-line is insecure. Consider using -P instead.
17/03/04 17:25:07 WARN sqoop.ConnFactory: Parameter --driver is set to an explicit driver however appropriate connection manager is not being set (via --connection-manager). Sqoop is going to fall back to org.apache.sqoop.manager.GenericJdbcManager. Please specify explicitly which connection manager should be used next time.
17/03/04 17:25:07 INFO manager.SqlManager: Using default fetchSize of 1000
17/03/04 17:25:07 INFO tool.CodeGenTool: Beginning code generation
```

```
17/03/04 17:25:09 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM dbo.reports AS t WHERE 1=0
17/03/04 17:25:10 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM dbo.reports AS t WHERE 1=0
17/03/04 17:25:11 INFO orm.CompilationManager: HADOOP_MAPRED_HOME is /usr/hdp/2.5.4.0-121/hadoop-mapreduce
Note: /tmp/sqoop-roby/compile/4d6a2c45fba5ef37eec34a7c2dd85f15/dbo_reports.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
17/03/04 17:25:13 INFO orm.CompilationManager: Writing jar file: /tmp/sqoop-
roby/compile/4d6a2c45fba5ef37eec34a7c2dd85f15/dbo.reports.jar
17/03/04 17:25:14 INFO mapreduce.ImportJobBase: Beginning import of dbo.reports
17/03/04 17:25:14 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM dbo.reports AS t WHERE 1=0
17/03/04 17:25:15 INFO impl.TimelineClientImpl: Timeline service address: http://headnodehost:8188/ws/v1/timeline/
17/03/04 17:25:16 INFO client.AHSProxy: Connecting to Application History server at headnodehost/10.0.0.18:10200
17/03/04 17:25:24 INFO db.DBInputFormat: Using read commited transaction isolation
17/03/04 17:25:24 INFO mapreduce.JobSubmitter: number of splits:1
17/03/04 17:25:25 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1488643045158_0005
17/03/04 17:25:29 INFO impl.YarnClientImpl: Submitted application application_1488643045158_0005
17/03/04 17:25:29 INFO mapreduce.Job: The url to track the job: http://hn0-
ireact.dz2inh1wxy1ubm3uqupbx44jtd.bx.internal.cloudapp.net:8088/proxy/application_1488643045158_0005/
17/03/04 17:25:29 INFO mapreduce.Job: Running job: job_1488643045158_0005
17/03/04 17:25:48 INFO mapreduce.Job: Job job_1488643045158_0005 running in uber mode : false
17/03/04 17:25:48 INFO mapreduce.Job:  map 0% reduce 0%
17/03/04 17:29:17 INFO mapreduce.Job:  map 100% reduce 0%
17/03/04 17:29:19 INFO mapreduce.Job: Job job_1488643045158_0005 completed successfully
17/03/04 17:29:19 INFO mapreduce.Job: Counters: 30
    File System Counters
        FILE: Number of bytes read=0
        FILE: Number of bytes written=171189
        FILE: Number of read operations=0
        FILE: Number of large read operations=0
        FILE: Number of write operations=0
        WASB: Number of bytes read=87
        WASB: Number of bytes written=812521275
        WASB: Number of read operations=0
        WASB: Number of large read operations=0
        WASB: Number of write operations=0
    Job Counters
        Launched map tasks=1
        Other local map tasks=1
        Total time spent by all maps in occupied slots (ms)=603516
        Total time spent by all reduces in occupied slots (ms)=0
        Total time spent by all map tasks (ms)=201172
        Total vcore-milliseconds taken by all map tasks=201172
        Total megabyte-milliseconds taken by all map tasks=309000192
    Map-Reduce Framework
        Map input records=1999999
        Map output records=1999999
        Input split bytes=87
        Spilled Records=0
        Failed Shuffles=0
        Merged Map outputs=0
        GC time elapsed (ms)=2529
        CPU time spent (ms)=39990
        Physical memory (bytes) snapshot=1223008256
        Virtual memory (bytes) snapshot=2838155264
        Total committed heap usage (bytes)=1048051712
    File Input Format Counters
        Bytes Read=0
    File Output Format Counters
        Bytes Written=812521275
17/03/04 17:29:19 INFO mapreduce.ImportJobBase: Transferred 0 bytes in 244.4043 seconds (0 bytes/sec)
17/03/04 17:29:19 INFO mapreduce.ImportJobBase: Retrieved 1999999 records.
```

5.1.2 Amazon EMR

All'interno dei servizi Amazon AWS sono necessarie alcune operazioni preliminari, in particolare è necessario garantire l'accesso ai servizi, siano essi RDS che EMR (Big Data) da parte della propria postazioni in uso e anche tra di loro (RDS ↔ EMR).
A tal fine è necessario modificare le impostazioni del Cluster EMR, partendo dall'elenco dei clusters:

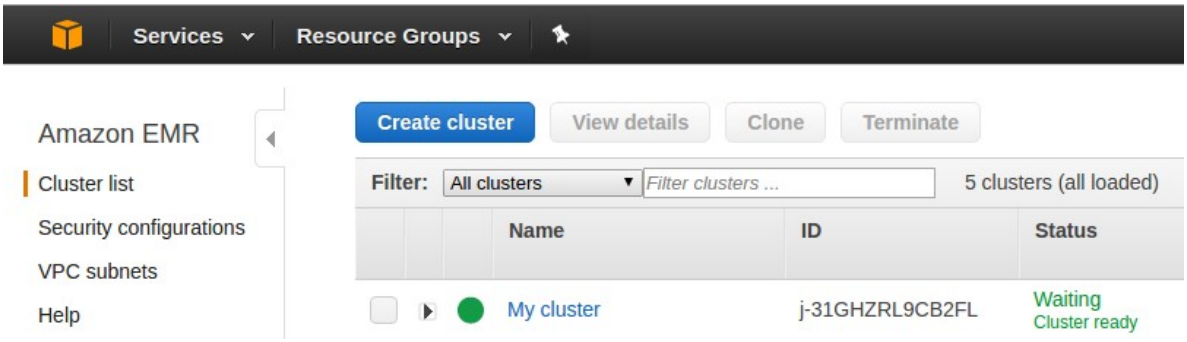


Figura 5.1 Elenco Cluster EMR

Scegliendo opportunamente il cluster creato e andando sulle impostazioni di sicurezza

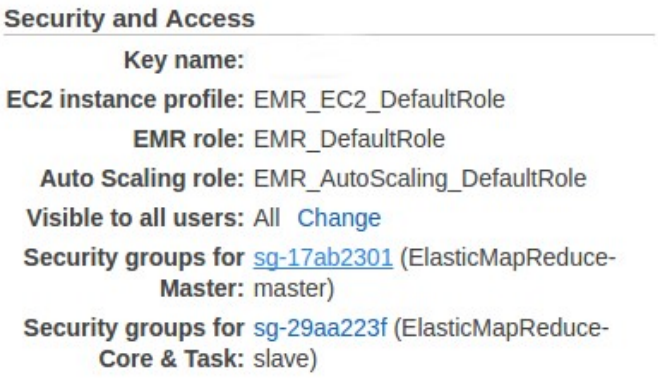


Figura 5.2 Impostazioni di Sicurezza Cluster EMR

Poi andando nelle impostazioni in Ingresso (INBOUND)

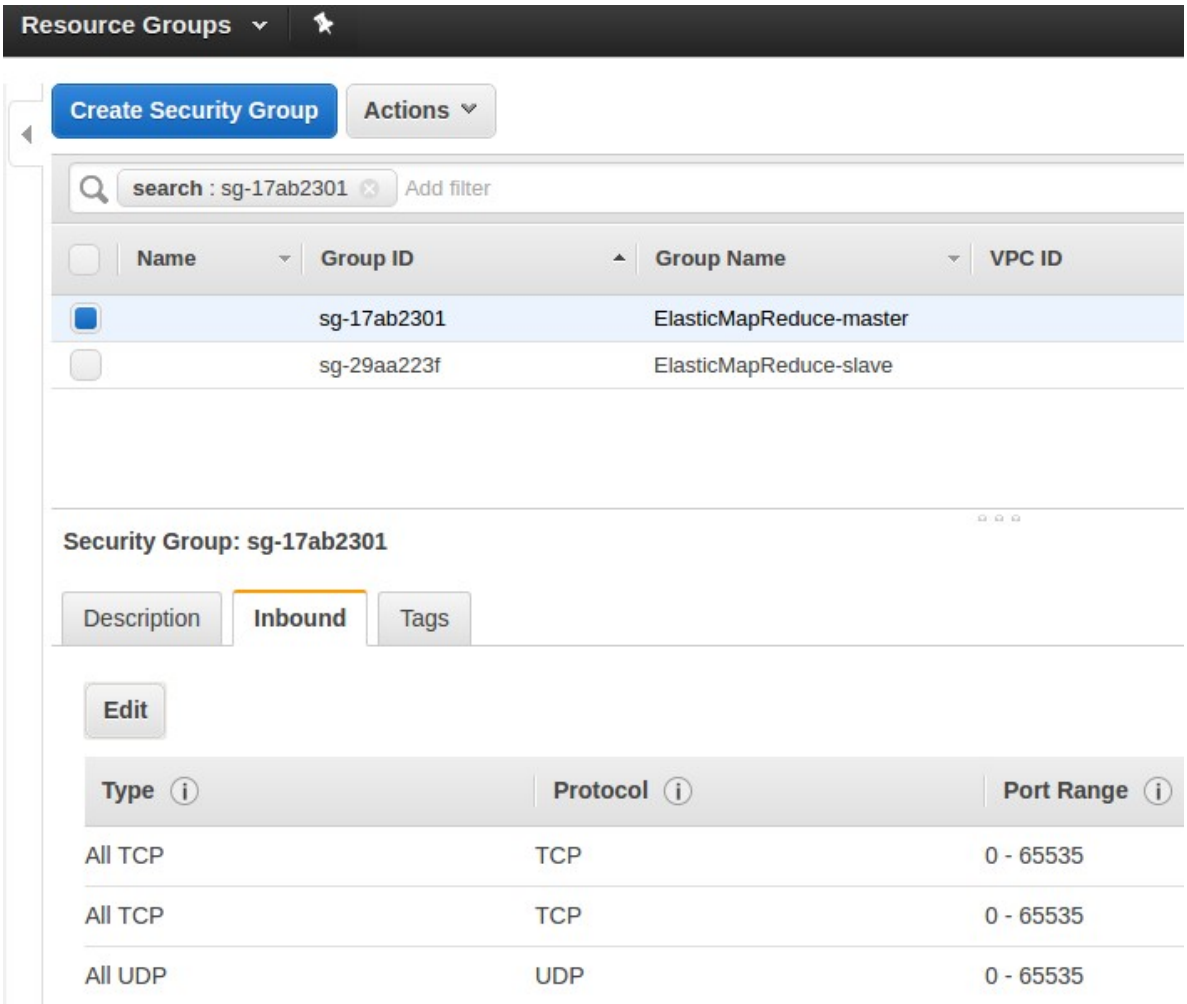


Figura 5.3 Security Group EMR

E aggiungendo la regola desiderata in base al proprio indirizzo IP di provenienza.

Custom TCP Rule	TCP	8443	Custom	54.240.217.16/29
Custom TCP Rule	TCP	8443	Custom	54.240.217.64/28
Custom TCP Rule	TCP	8443	Custom	54.240.217.8/29
Custom TCP Rule	TCP	8443	Custom	54.240.217.80/29
Custom TCP Rule	TCP	8443	Custom	213.254.0.0/32
Custom TCP Rule	TCP	8443	Custom	213.254.0.0/24
Custom TCP Rule	TCP	8443	Custom	213.254.0.0/16
Custom TCP Rule	TCP	8443	Custom	213.254.0.0/15
Custom TCP Rule	TCP	0	Custom	213.254

Add Rule

Figura 5.4 Firewall Rules per EMR

Lo stesso tipo di discorso vale anche per il servizio RDS, che richiede la configurazione delle regole per permettere ai client di collegarsi (si tenga presente che l'applicativo Sqoop è anch'esso un client SQL Server).

Security Group: sg-77886408

Description | Inbound | Outbound | Tags

Edit

Type	Protocol	Port Range	Source
MS SQL	TCP	1433	0.0.0.0/0
MS SQL	TCP	1433	::/0

Figura 5.5 Security Group per RDS

Per quanto riguarda i test di esportazione da RDBMS, non esiste un tool come il Data Factory all'interno dei servizi Amazon, quindi verrà riportato direttamente il risultato dell'esecuzione del comando Sqoop:

Comando Sqoop

sqoop import --connect "jdbc:sqlserver://ireact.cfqkxv1xqnfp.us-east-1.rds.amazonaws.com;database=ireact" --username admin --password <password> --driver com.microsoft.sqlserver.jdbc.SQLServerDriver --table "dbo.reports" --target-dir output --m 1;

output

Tempo impiegato su Amazon: 1 minuto

```
[hadoop@ip-10-31-110-197 ~]$ date> data;sqoop import --connect "jdbc:sqlserver://ireact.cfqkxv1xqnfp.us-east-1.rds.amazonaws.com;database=ireact" --username admin --password <password> --driver com.microsoft.sqlserver.jdbc.SQLServerDriver --table "dbo.reports" --target-dir output --m 1;date >> data
Warning: /usr/lib/sqoop/../accumulo does not exist! Accumulo imports will fail.
Please set $ACCUMULO_HOME to the root of your Accumulo installation.
17/03/05 12:44:56 INFO sqoop.Sqoop: Running Sqoop version: 1.4.6
17/03/05 12:44:56 WARN tool.BaseSqoopTool: Setting your password on the command-line is insecure. Consider using -P instead.
17/03/05 12:44:56 WARN sqoop.ConnFactory: Parameter --driver is set to an explicit driver however appropriate connection manager is not being set (via --connection-manager). Sqoop is going to fall back to org.apache.sqoop.manager.GenericJdbcManager. Please specify explicitly which connection manager should be used next time.
```

```
17/03/05 12:44:56 INFO manager.SqlManager: Using default fetchSize of 1000
17/03/05 12:44:56 INFO tool.CodeGenTool: Beginning code generation
17/03/05 12:44:57 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM dbo.reports AS t WHERE 1=0
17/03/05 12:44:57 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM dbo.reports AS t WHERE 1=0
17/03/05 12:44:57 INFO orm.CompilationManager: HADOOP_MAPRED_HOME is /usr/lib/hadoop-mapreduce
Note: /tmp/sqoop-hadoop/compile/8d830e6f0fa7725c11b2d80b3fa469b8/dbo_reports.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
17/03/05 12:45:00 INFO orm.CompilationManager: Writing jar file: /tmp/sqoop-hadoop/compile/8d830e6f0fa7725c11b2d80b3fa469b8/dbo.reports.jar
17/03/05 12:45:00 INFO mapreduce.ImportJobBase: Beginning import of dbo.reports
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/hadoop/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/hive/lib/log4j-slf4j-impl-2.4.1.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
17/03/05 12:45:01 INFO Configuration.deprecation: mapred.jar is deprecated. Instead, use mapreduce.job.jar
17/03/05 12:45:01 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM dbo.reports AS t WHERE 1=0
17/03/05 12:45:01 INFO Configuration.deprecation: mapred.map.tasks is deprecated. Instead, use mapreduce.job.maps
17/03/05 12:45:02 INFO impl.TimelineClientImpl: Timeline service address: http://ip-10-31-110-197.ec2.internal:8188/ws/v1/timeline/
17/03/05 12:45:02 INFO client.RMProxy: Connecting to ResourceManager at ip-10-31-110-197.ec2.internal/10.31.110.197:8032
17/03/05 12:45:08 INFO db.DBInputFormat: Using read commited transaction isolation
17/03/05 12:45:09 INFO mapreduce.JobSubmitter: number of splits:1
17/03/05 12:45:09 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1488715174821_0001
17/03/05 12:45:09 INFO impl.YarnClientImpl: Submitted application application_1488715174821_0001
17/03/05 12:45:09 INFO mapreduce.Job: The url to track the job: http://ip-10-31-110-197.ec2.internal:20888/proxy/application_1488715174821_0001/
17/03/05 12:45:09 INFO mapreduce.Job: Running job: job_1488715174821_0001
17/03/05 12:45:19 INFO mapreduce.Job: Job job_1488715174821_0001 running in uber mode : false
17/03/05 12:45:19 INFO mapreduce.Job:  map 0% reduce 0%
17/03/05 12:45:55 INFO mapreduce.Job:  map 100% reduce 0%
17/03/05 12:45:56 INFO mapreduce.Job: Job job_1488715174821_0001 completed successfully
17/03/05 12:45:56 INFO mapreduce.Job: Counters: 30
    File System Counters
        FILE: Number of bytes read=0
        FILE: Number of bytes written=147223
        FILE: Number of read operations=0
        FILE: Number of large read operations=0
        FILE: Number of write operations=0
        HDFS: Number of bytes read=87
        HDFS: Number of bytes written=812521826
        HDFS: Number of read operations=4
        HDFS: Number of large read operations=0
        HDFS: Number of write operations=2
    Job Counters
        Launched map tasks=1
        Other local map tasks=1
        Total time spent by all maps in occupied slots (ms)=1505790
        Total time spent by all reduces in occupied slots (ms)=0
        Total time spent by all map tasks (ms)=33462
        Total vcore-milliseconds taken by all map tasks=33462
        Total megabyte-milliseconds taken by all map tasks=48185280
    Map-Reduce Framework
        Map input records=1999998
        Map output records=1999998
        Input split bytes=87
        Spilled Records=0
        Failed Shuffles=0
        Merged Map outputs=0
        GC time elapsed (ms)=170
        CPU time spent (ms)=38420
        Physical memory (bytes) snapshot=547319808
        Virtual memory (bytes) snapshot=3195457536
        Total committed heap usage (bytes)=499122176
    File Input Format Counters
        Bytes Read=0
```

```
    File Output Format Counters
        Bytes Written=812521826
17/03/05 12:45:56 INFO mapreduce.ImportJobBase: Transferred 774,8812 MB in 54,6218 seconds (14,1863 MB/sec)
17/03/05 12:45:56 INFO mapreduce.ImportJobBase: Retrieved 1999998 records.
```

Come si può osservare dal Log, il tempo richiesto per l'esportazione è di circa 55 secondi, notevolmente più basso rispetto a quello di Azure.

5.2 Strumenti e fonti dati GeoReferenziali

Per poter effettuare dei test interessanti è importante prima avere a disposizione dei set di dati in ambito georeferenziale, che abbiamo una qualche rappresentazione e significato. A tal fine, nelle prossime sezioni, verranno esaminati una serie di siti da cui reperire informazioni preziose per i test.

5.2.1 Dati geografici - NaturalEarthData

I dati geografici, quali:

- Confini territoriali,
- limiti di fiumi e mari
- centri e costruzioni urbane
- altre tipologie

possono essere recuperati da diversi siti che raccolgono ed aggiornano queste informazioni.

Un sito molto comodo è http://www.naturalearthdata.com: i dati sono liberamente scaricabili senza alcuna limitazione, come specificato dalle note presenti sul sito:

All versions of Natural Earth raster + vector map data found on this website are in the public domain. You may use the maps in any manner, including modifying the content and design, electronic dissemination, and offset printing. The primary authors, *Tom Patterson and Nathaniel Vaughn Kelso*, and all other contributors renounce all financial claim to the maps and invites you to use them for personal, educational, and commercial purposes.

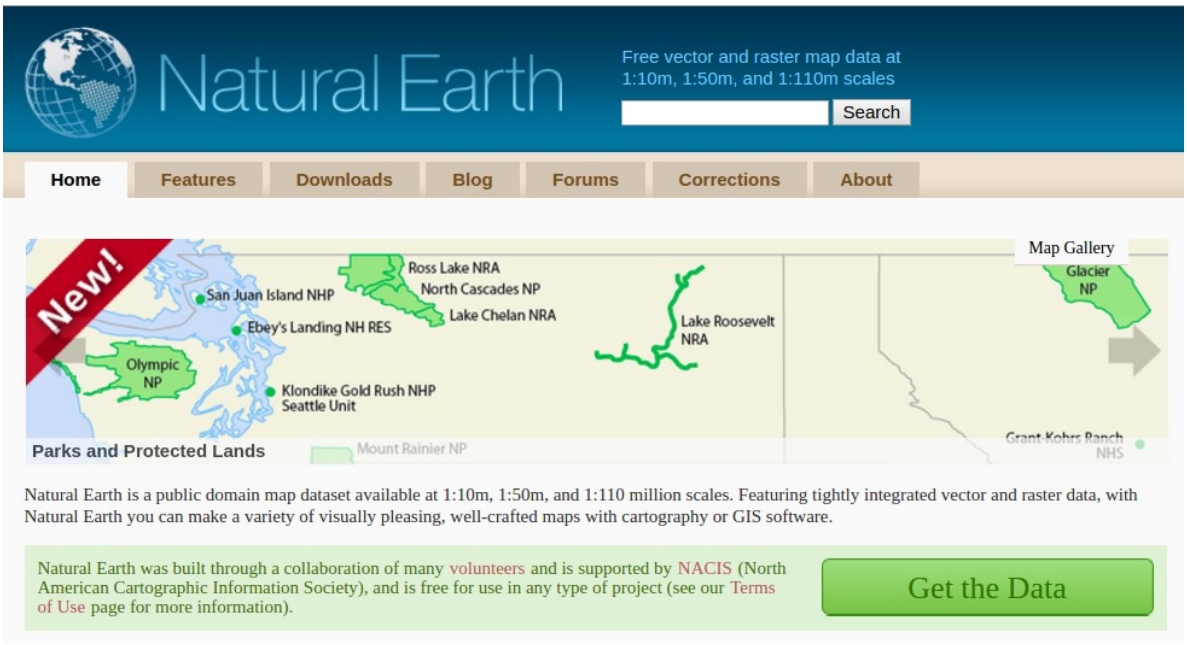


Figura 5.6 Sito Web Natural Earth Data

I dati sono disponibile a 3 scale differenti:

- 1:10.000.000
- 1:50.000.000
- 1:110.000.000

Per ogni scala, sono disponibili dati di tipo:

- Culturale (ad esempio confini di paesi, strade, ferrovie, aeroporti, parchi, fusi orari, edifici, ecc.)
- Fisici (coste, territori, oceani, fiumi, laghi, spiagge, zone ipobarometriche, linee geografiche, reticoli di coordinate, ecc.)
- Dati da immagini satellitari con rilievi geografici.

Una volta scaricati i dati è necessario effettuare delle conversioni, a partire dal formato che viene utilizzato: in particolare, la maggiorparte di siti utilizza il formato standard shapefile ESRI che permette di specificare dei "vettori con attributi", dove per vettori si intende:

- punti,
- linee,
- poligoni

mentre per gli attributi si intendono ad esempio:

- temperature,
- pressione,
- intensità di un terremoto (magnitudo – scala Richter)

Il formato shapefile che usa l'estensione .shp, è un formato binario che richiede l'utilizzo di un applicativo per poter estrarne i dati.

5.2.2 Dati geografici – Geofabrik

Un altro sito interessante è: https://www.geofabrik.de/

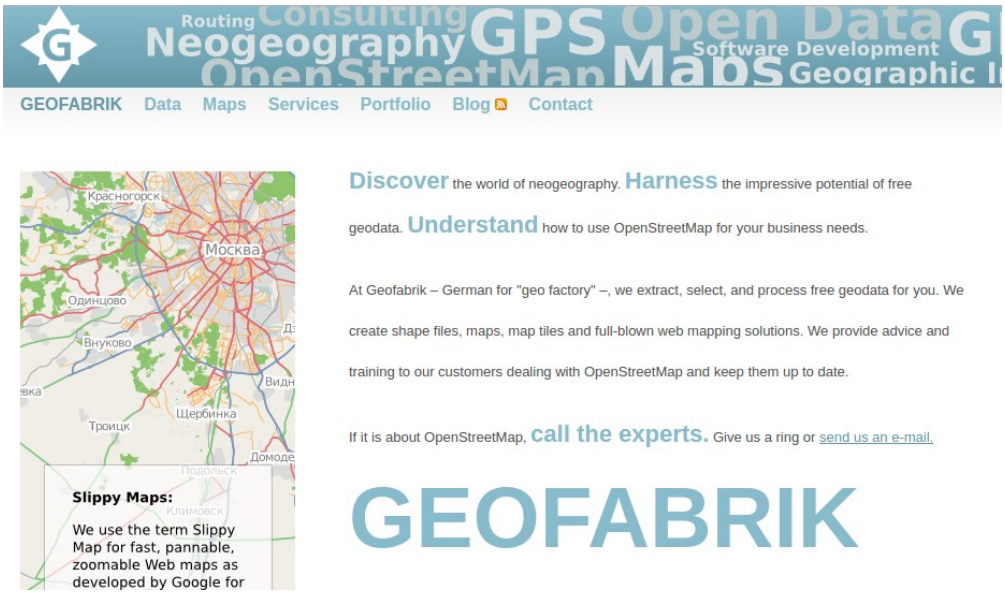


Figura 5.7 Sito Web GeoFabrik

Questo sito raccoglie una grandissima quantità di dati, con una precisione molto elevata.

Nella figura è riportata una rappresentazione tramite QGIS di alcuni dati presi da GeoFabrik, che mostrano diversi tipi di oggetti (strutture, edifici, fiumi) a Torino

Figura 5.8 Esempio tramite QGis di dati presi da GeoFabrik

5.2.3 Dati Eventi Sismici mondiali

Per quanto riguarda, ad esempio, gli eventi sismici mondiali, esistono dei database contenenti informazioni molto precise.
Uno dei siti più famosi è http://www.emsc-csem.org

Figura 5.9 Sito Web Eventi Sismici mondiali

E' possibile scaricare l'elenco dei dati relativi alle scosse sismiche dal 2004 ad oggi, indicando:

- La zona geografica precisa (longitudini e latitudini)
- Periodo temporale
- Attributi aggiuntivi (presenza di immagini o commenti)

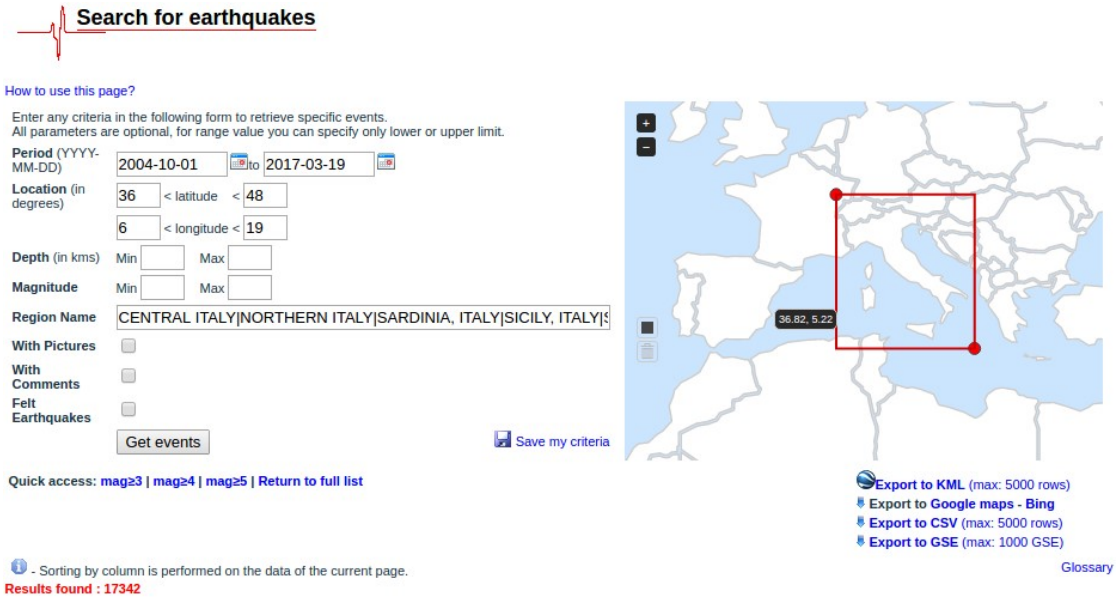


Figura 5.10 Esempio esportazione dati da Sito Web www.emsc-csem.org

Nota: è possibile scaricare al massimo 5000 righe, quindi 5000 posizioni geografiche.

Per ogni record vengono mostrati:

- longitudine e latitudine
- Profondità della scossa (ipocentro terremoto)
- Magnitudo (grado scala Richter)
- Regione Geografica.

5.2.4 Applicativi per la manipolazione dei dati geografici

Tra i vari software disponibili sul mercato, per la gestione di dati geografici, è presente la categoria denominata "GIS" Geographic Information System: in particolare è molto famoso il Software QGIS, un Sistema di Informazione Geografica Libero e Open Source, sito web https://www.qgis.org/it/site/

QGIS è un software facile da usare e multi piattaforma, funzionando su ambienti:

- Microsoft Windows
- Mac OS X
- Linux
- BSD
- Android

Figura 5.11 Logo QGIS

Un volta installato il software, e una volta scaricata la mappa, occorrerà aprire la mappa, aggiungendo un nuovo vettore

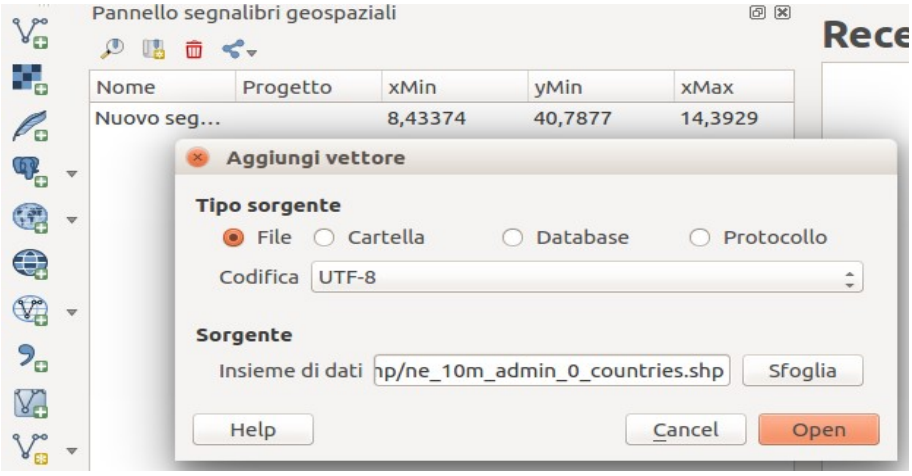


Figura 5.12 Aggiunta Vettore su QGIS

Nella

figura che mostra tutto il mondo, si seleziona il paese (o i paesi) interessati, cliccando 2 volte sull'area relativa al paese.

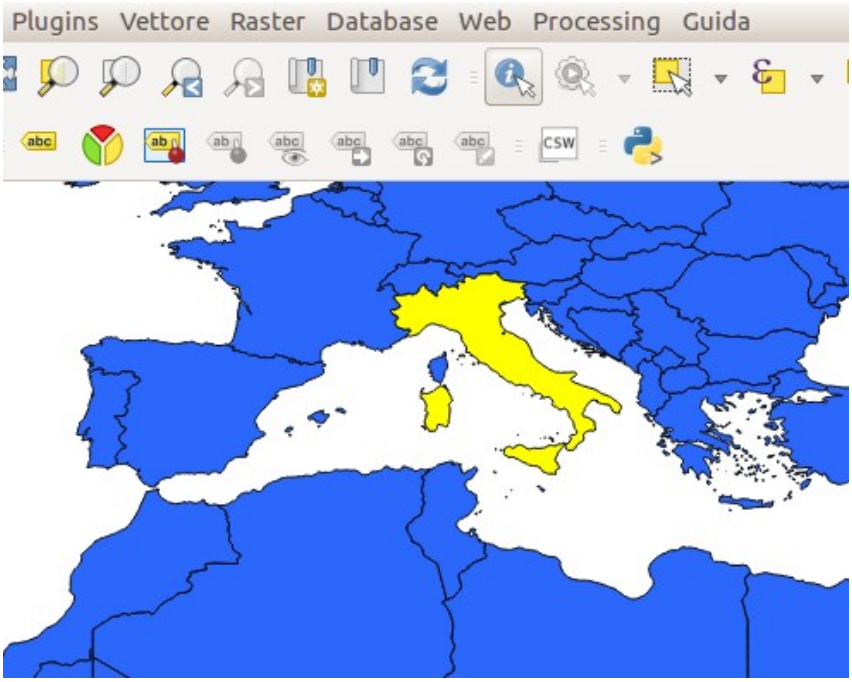


Figura 5.13 Selezione zona da esportare su QGIS

Dopo di ché bisogna aprire il pannello informazioni sulla destra (si veda figura): cliccando nuovamente sull'area selezionata, verranno selezionati tutti gli oggetti relativi all'area

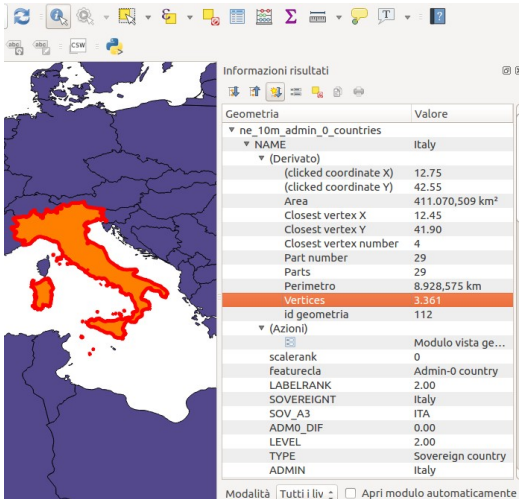


Figura 5.14 Selezione oggetti interni alla zona selezionata

A questo punto, basterà esportare i dati, cliccando su “Salva con nome” e scegliendo il formato desiderato (GeoJSON o .CSV)

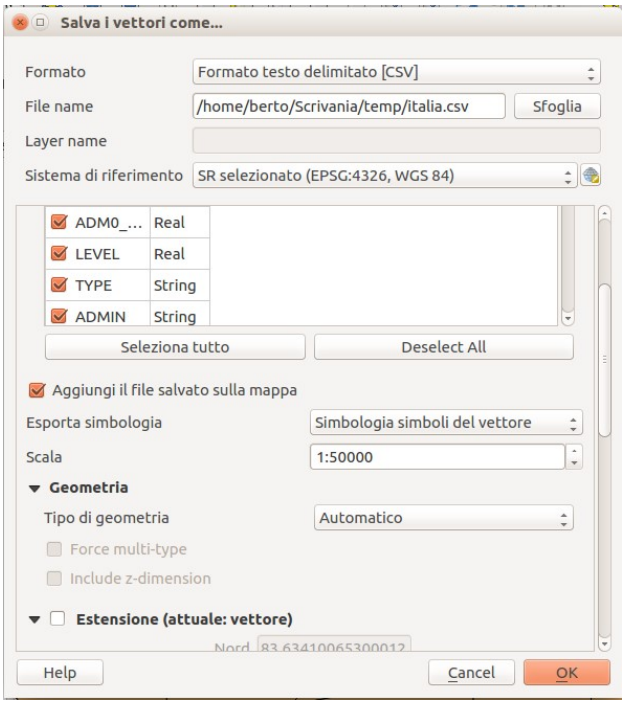


Figura 5.15 Esportazione vettore QGIS

5.2.5 Elaborazione dati georeferenziali

Una volta che abbiamo a disposizione i dati da analizzare ed elaborare, non manca che scrivere il codice Spark.

Prima di tutto dobbiamo identifica la zona di interesse (che poi sarà quella su cui si costruirà l’immagine finale).

Per far questo utilizziamo, a livello approssimativo, il rettangolo con:

- 6 < longitudine < 19
- 36 < latitudine < 48

L’istruzione corrispondente è:

Envelope itArea = new Envelope(6,19,36,48);

Come seconda istruzione, è importante mostrare il reticolato relativo alle principali latitudini e longitudini; per fare questo definiamo un nuovo metodo, "creaLinea" che prende in input la coordinata iniziale, quella finale, e le 2 risoluzioni (cioè la densità dei punti, la prima per la longitudine, la seconda per la latitudine) intermedi tra le 2 coordinate.
Il metodo risulta così essere:

```
public static ArrayList <Point> creaLinea(double x0, double y0, double x1, double y1, double resolutionX, double resolutionY) {
	GeometryFactory gf = new GeometryFactory();
	ArrayList <Point> punti = new ArrayList<Point>();
	for (double x=x0; x <= x1; x += resolutionX) {
		for (double y=y0; y <= y1; y += resolutionY) {
			punti.add(gf.createPoint( new Coordinate( x, y)));
		}
	}
	return punti;
}
```

Il metodo restituisce un'array contenente tutti i punti del segmento della linea:

```
ArrayList <Point> listaDirettrici = new ArrayList<Point>();
for (int lo=6; lo<19; lo += 3)
	for (int la=36; la<=48; la += 3) {
		listaDirettrici.addAll(creaLinea(lo,la, lo, la+3, prec, prec));
		listaDirettrici.addAll(creaLinea(lo,la, lo+3, la, prec, prec));
	}
```

Un'altra istruzione comoda, può essere quella di rappresentare il centro di una città, ad esempio Torino, tramite un nuovo metodo, la creaQuadratoPieno che, data una coordinata, 2 delta (x ed y) e 2 risoluzioni (x ed y), restituisce un array con tutti i punti creati per formare un quadrato che ha come centro la coordinata iniziale e come mazzi lati i 2 delta passati in input.
Il codice del metodo creaQuadrataPieno è:

```
public static ArrayList <Point> creaQuadratoPieno(double x0, double y0, double deltaX, double deltaY, double resolutionX, double resolutionY) {
	GeometryFactory gf = new GeometryFactory();
	ArrayList <Point> punti = new ArrayList<Point>();
	for (double x=x0-deltaX; x < x0+deltaX; x += resolutionX) {
		for (double y=y0-deltaY; y < y0+deltaY; y += resolutionY) {
			punti.add(gf.createPoint( new Coordinate( x, y)));
		}
	}
	return punti;
```

```
}
```

Mentre il codice utilizzato per creare il quadrato di Torino è:

```
// Torino
double prec = 0.01;
ArrayList <Point> listaTorino = new ArrayList<Point>();
GeometryFactory gf = new GeometryFactory();
listaTorino.addAll(creaQuadratoPieno(7.6868, 45.07, 0.1, 0.1, prec, prec)); // Torino
```

Dal sito NaturalEarthData si può scaricare un elenco di punti che rappresentano il confine Italiano: è possibile sfruttare il metodo appena visto creaQuadratoPieno, per "inspessire" le coordinate dei confini geografici, per renderli più evidenti nell'immagine che si vedrà alla fine.

Per ottenere lo scopo, quindi, prima si leggono le coordinate da un file CSV, poi si effettua la conversione da JAVARDD <String> a JAVARDD <Point>.
Successivamente si utilizza il metodo "flatMap" che si aspetta come parametro una classe che implementi l'interfaccia FlatMapFunction che, dato un punto, restituisce un insieme di punti.

Segue il codice relativo:

```
JavaRDD<String> italiaString = sc.textFile("in/italy.csv");
JavaRDD<Point> italia = convertiInJavaRDDPoint(italiaString);
JavaRDD<Point> italia_spessa = italia.flatMap(new FlatMapFunction<Point,Point>() {
	public Iterable<Point> call(Point t) throws Exception {
		double prec = 0.01;
		double delta = 0.01;
		return creaQuadratoPieno(t.getX(),t.getY(), delta, delta, prec, prec);

	}
});
PointRDD puntiRDDItalia = new PointRDD(italia_spessa);
```

Per quanto riguarda i dati relativi alle scosse di terremoto, si è pensato di applicare lo stesso concetto visto sopra, creando per ogni punto, un'insieme di punti determinati da un quadrato: in questo caso però, invece di utilizzare un grandezza del quadrato costante, si è deciso di renderlo proporzionale alla magnitudo del terremoto; in questo modo si avrà un'area di influenza più attinente alla realtà e al tipo di terremoto.

Il codice del metodo è:

```
private static JavaRDD <Point> convertiInJavaRDDPointScossePesate(JavaRDD<String> rdd) {
	// Calcola coordinate
	JavaRDD <Point> javaRDDPoint = rdd.flatMap(new FlatMapFunction<String,Point>() {
		public Iterable<Point> call(String row) {
			String aPoint[] = row.split(";");
			double longitude = new Double(aPoint[3]);
			double latitude = new Double(aPoint[2]);
			//double depth = new Double(aPoint[4]);
			double magnitude = new Double(aPoint[7]);

			GeometryFactory gf = new GeometryFactory();
			Point t = gf.createPoint( new Coordinate( longitude, latitude) );
			double factor = 10000;
			double prec = 1/factor;
			double delta = Math.pow(10,magnitude/2)/factor;
			return creaQuadratoPieno(t.getX(),t.getY(), delta, delta, prec, prec);
		}
	});
	return javaRDDPoint;
}
```

Il codice che utilizza il metodo è:

```
// Scosse
JavaRDD<String> scosseString = sc.textFile("in/scosse-it.csv");
JavaRDD<Point> scosse = convertiInJavaRDDPointScossePesate(scosseString);
```

E' importante, una volta effettuate le elaborazioni di "inspessimento", convertire il dato da JAVARDD<Point> a PointRDD, espressamente richiesto dalle librerie grafiche GeoSpark:

```
PointRDD puntiRDDItalia = new PointRDD(italia_spessa);
PointRDD puntiRDDTorino = new PointRDD(puntiTorino);
PointRDD puntiRDDDirettrici = new PointRDD(puntiDirettrici);
PointRDD puntiRDDScosse = new PointRDD(scosse);
```

Come pezzo finale, viene richiamato il metodo di creazione immagine heatMap:

```
public static  void heatMap(JavaSparkContext sc, SpatialRDD oggetti,String nomeFile, Envelope area) throws
Exception {
	HeatMap visualizationOperator = new HeatMap(height,width,area,false,3);
	visualizationOperator.Visualize(sc, oggetti);
	NativeJavaImageGenerator imageGenerator = new  NativeJavaImageGenerator();
	imageGenerator.SaveAsFile(visualizationOperator.pixelImage,"target/heatmap/"+nomeFile,
ImageType.PNG);
}
```

che, come visto in precedenza, si occupa di creare un'immagine simile ad una "mappa di calore" in base alla densità dei punti passati in input.

Come istruzioni finali, viene richiamata la heatMap, per ogni oggetto analizzato in precedenza:

- Torino
- Reticoli Longitudini e Latitudini
- Confine Italiano
- Scosse Terremoti

```
try {
	heatMap(sc,puntiRDDTorino,"Torino",itArea);
	heatMap(sc,puntiRDDDirettrici,"Direttrici",itArea);
	heatMap(sc,puntiRDDScosse,"Scosse",itArea);
	heatMap(sc,puntiRDDItalia,"Italia",itArea);
} catch(Exception e) {
	System.out.println("Errore: "+e);
};
```

Ogni Immagine può essere generata ed elaborata singolarmente: utilizzando un elaboratore di immagini come Gimp, è possibile estrarre ogni immagine e sovrapporla ad uno sfondo nero.

Il risultato finale è visible nella figura 5.16

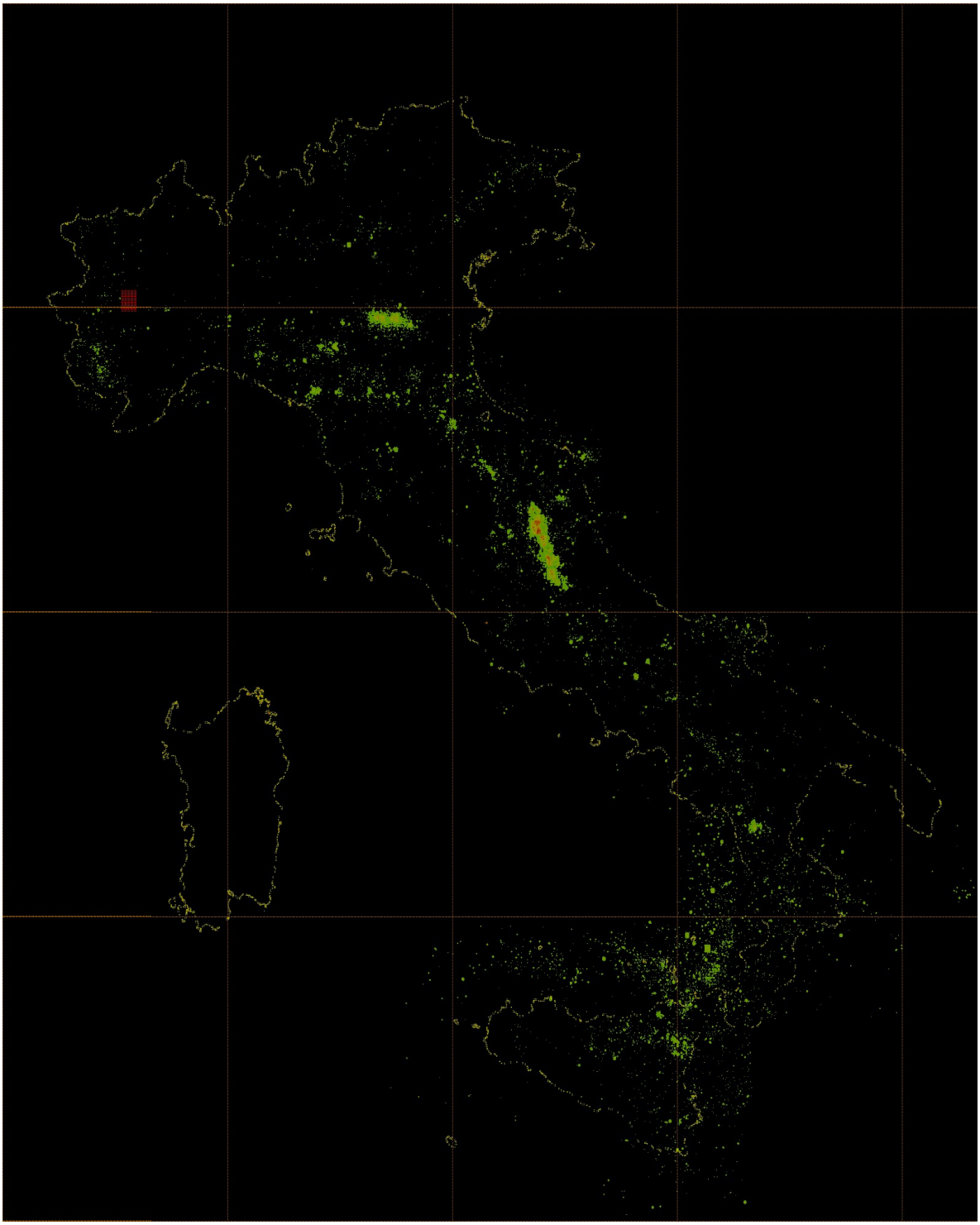


Figura 5.16 Immagine finale prodotto alla fine dell'elaborazione di tutti gli oggetti georeferenziati

5.2.6 Note sull'esecuzione in Cluster

Per poter eseguire il programma Spark su ambiente Cluster sono necessarie alcune piccole accortezze:

- E' importante verificare che i dati in input (in particolare le coordinate e relativi attributi) siano accessibili tramite file system HDFS.
  Per far questo bisognerà preventivamente copiare tutta la directory "in", contenente i dati in formato .CSV, su HDFS tramite il comando:
  hdfs dfs -put in
- Per elaborazioni che coinvolgono grandi moli di dati, sarà necessario poter usufruire di una adeguata quantità di memoria centrale: a tal fine bisognerà modificare lo script di lancio di Spark, aggiungendo il parametro:
  --driver-memory 20g
  dove 20g significa 20 GigaByte di RAM utilizzabile.
  Nota: è necessario, in fase di partenza del job Spark, che il messaggio seguente mostri la corretta dimensione di memoria allocabile:
  INFO Client: Will allocate AM container, with 1408 MB memory including 384 MB overhead.
  Al posto di 1408 dovrà comparire il valore specificato, espresso in MB (nell'esempio 20480).
- La modalità di lancio del JOB sarà diversa dal caso standalone, grazie ai parametri:
  --deploy-mode cluster
  che specificherà la modalità "Cluster"
  --master yarn
  che dirà di utilizzare yarn e hdfs.

Capitolo 6 Analisi Costi

6.1 Costi Amazon AWS

I prezzi relativi ai servizi Amazon sono consultabili tramite il link: https://aws.amazon.com/it/ec2/pricing/

Esistono molte possibilità di configurazione di Cluster, dal più economico al più performante: si passa 0,011 USD a 0,27 USD all'ora per istanza in 1 ora
Il tipo di fatturazione, in generale, di Amazon, è totalmente bilanciato; questo vuol dire che, ad esempio 1 macchina con 10 istanze attive per 10 ore equivalgono allo stesso costo di 100 istanze attive per 1 ora.

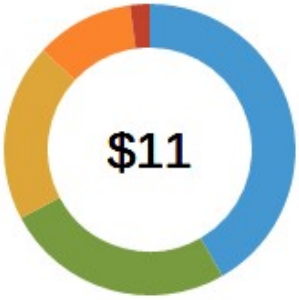
Per quanto riguarda i servizi RDS, quindi Database Server, i prezzi vanno dai 0.022$ /ora a 38.542$ /ora a seconda del tipo di servizio, di location e di server.

6.1.1 Esempio di utilizzo servizi AWS

Segue un'immagine riepilogativa dei costi, con relativi dettagli del tipo di servizio usato.
I servizi offerti da Amazon presentano una fatturazione molto dettagliata. Distinguendo tra:

- EC2, Elastic Compute Cloud che rappresenta la risorsa base per l'elaborazione (quindi un elaboratore con RAM, CPU e DISCO). Il termine "Elastic" sotto intende il fatto che ogni macchina può essere accesa e spenta quando si vuole, pagando soltanto le ore di attività della macchina stessa.
- RDS rappresenta, in generale, l'interfaccia verso gli ambienti Database Sql
- Database Migration Svc, riguarda la comunicazione tra database e mondo esterno
- Elastic Map Reduce sono i servizi di EMR, quindi Hadoop e Spark: in particolare, per il caso in oggetto, Spark

The chart below shows the proportion of costs spent for each service you use.



EC2	$4.02
RDS	$2.46
DatabaseMigrationSvc	$1.90
ElasticMapReduce	$1.05
Other Services	$0.21
Tax	$2.09
Total	**$11.73**

Figura 6.1 Esempio Fatturazione Servizi Amazon AWS

6.2 Costi Microsoft Azure

Per dare un'occhiata ai costi dell'infrastruttura Microsoft Azure si può visitare il sito https://azure.microsoft.com/it-it/pricing/

I servizi relativi al cluster HDInsight vanno dai 0,073$ /ora per ogni nodo a 2,24$ /ora per ogni nodo.

Per quanto riguarda i servizi di Database SqlServer, i prezzi vanno da 0,091$ /ora a 24,08$ / ora, a secondo del tipo di servizio.

Segue un esempio di fatturazione servizi Azure

NOME	TIPO	G.	SPESA (EUR)
cluster-2017	Cluster HDInsight	...	17,15
cluster-ireact2017	Cluster HDInsight	...	16,02
fscamsqlSrv14	Macchina virtuale	...	11,65
fscamwebSrv1	Macchina virtuale	...	5,83
fscamwebSrv0	Macchina virtuale	...	5,83
cluster-2016	Cluster HDInsight	...	5,77
ireact-spark1	Cluster HDInsight	...	5,36
reactdb	SQL Server	...	3,70
fscamstgurtwe7wipl2ks	Account di archiviazione	...	3,62
fscamsqlSrv14	Macchina virtuale	...	1,56

Figura 6.2 Esempio fatturazione servizi Microsoft Azure

Capitolo 7 Note Conclusive

Il lavoro svolto nella seguente tesi mette in evidenza una nuova architettura utilizzabile per gestire dati georeferenziali e non solo; in particolare i dati posizionali possono essere messi in relazione con altri tipi di dati (o attributi) come, ad esempio, magnitudo di scosse sismiche, temperatura, pressione o altri dati aggiuntivi alla posizione.

L'utilità di questa architettura può essere sfruttata al meglio nel caso di elevate moli di dati (come accennato all'inizio, ordine di grandezza dei Peta Bytes, 1PB = 10^{15} Bytes): per questo motivi, l'architettura Big Data (soprattutto che tratta dati georeferenziati) è particolarmente adatta a gestire dati relativi ad utenti di Social Network (Twitter, Facebook, WhatsApp, Instagram, ecc.) per i più svariati scopi:

1. Ricerche di mercato (a scopo commerciale)
2. Analisi fenomeni di tendenza sociale
3. Analisi dati rivelabili da sensori (traffico, inquinamento, scosse sismiche, ecc.)

Nella presente trattazione si è cercato di fornire gli strumenti per poter comprendere ed utilizzare l'architettura Big Data, di intuirne le potenzialità e di prendere dimestichezza con gli strumenti più diffusi a disposizione (Cloud Providers).

Un approfondimento che non è stato possibile trattare e che, può essere oggetto di un ulteriore approfondimento, è un'analisi dettagliata delle tempistiche e delle risorse utilizzate (e quindi dei costi richiesti) per poter eseguire determinate elaborazioni. Particolarmente interessante sarebbe la valutazione delle capacità di "Scaling" dei Cloud Providers, quindi poter verificare come sfruttare al meglio i servizi OnDemand per minimizzare i costi e massimizzare i risultati attesi.

Un altro campo di approfondimento potrebbe essere, nell'ambito degli eventi sismici, mettere in relazione l'intensità e la posizione dei terremoti con la disposizione di strutture ed edifici artificiali o con la morfologia del territorio (ad esempio analizzando i possibili impatti di terremoti a elevata magnitudo su zone fluviali per poter prevedere l'entità di un evento sismico).

Capitolo 8 Dettagli su Microsoft Azure

8.1 Utilizzo del Data Factory

L'ambiente Cloud di Azure mette a disposizione una serie di strumenti grafici per semplificare le attività di migrazione dati da un ambiente all'altro; lo stumento specifico utilizzato è stato il Data Factory.

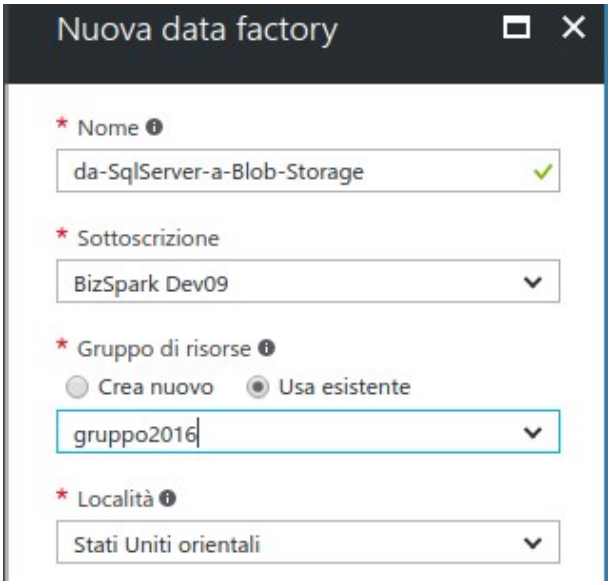


Figura 8.1 Creazione Data Factory

Durante la creazione del Data Factory bisognerà indicare il nuovo nome, l'account, il gruppo di risorse (che deve essere stato precedentemente creato) e la località intesa come locazione fisica del Data Center da utilizzare.

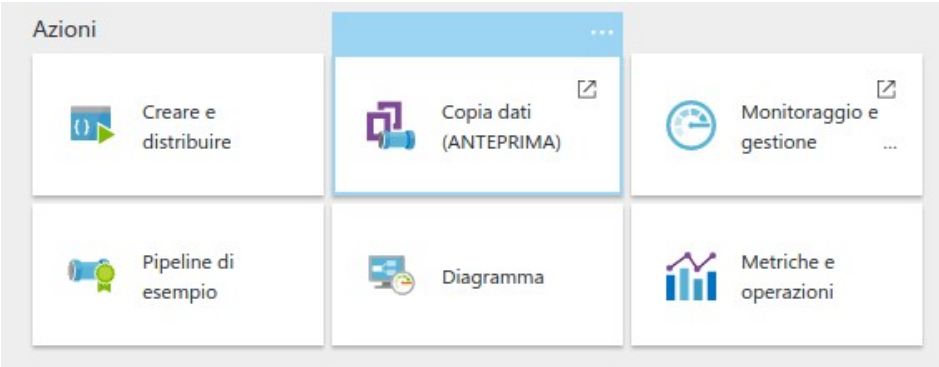


Figura 8.2 Menu Data Factory

Una volta creata, bisognerà andare nella voce relativa al nuovo DataFactory e cliccare su Copia Dati per accedere al Wizard di esportazione dati.

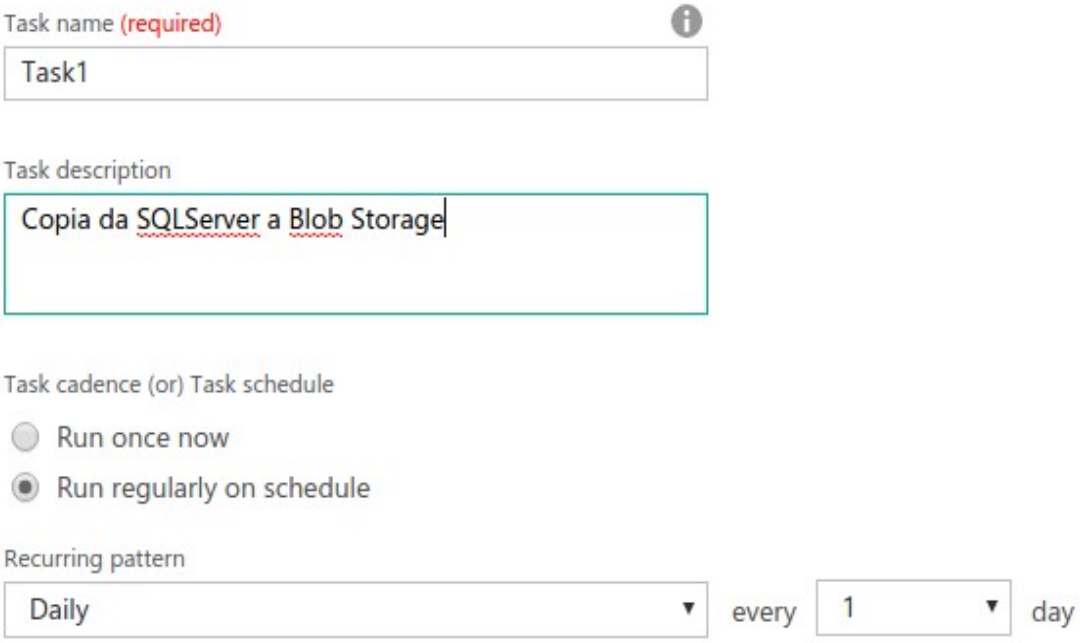


Figura 8.3 Creazione Pipeline

Nelle proprietà bisognerà impostare il nome del Task, la descrizione e, soprattutto il tipo di schedulazione: quest'ultima potrà essere immediata oppure schedulata per essere eseguita con una determinata cadenza, ad esempio una volta al giorno.

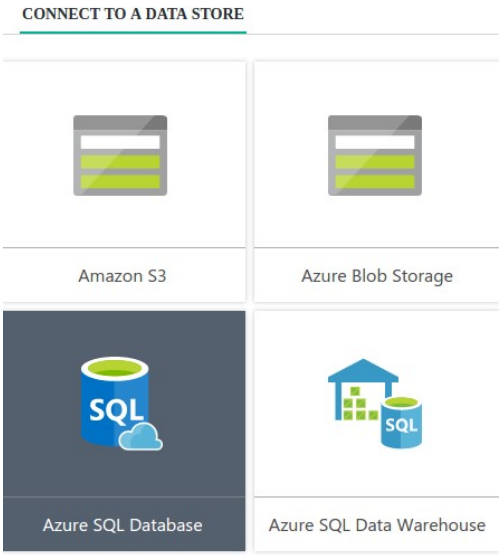


Figura 8.4 Selezione Data Store Sorgente

Come connessione al Data Store specifichiamo l'RDBMS e forniamo i dettagli per la connessione.

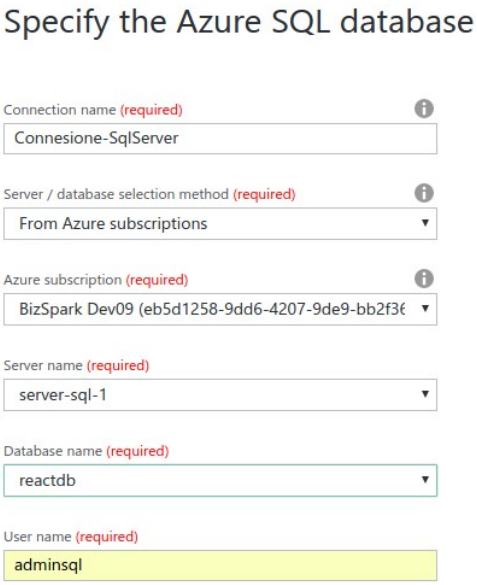


Figura 8.5 Selezione Connessione Database

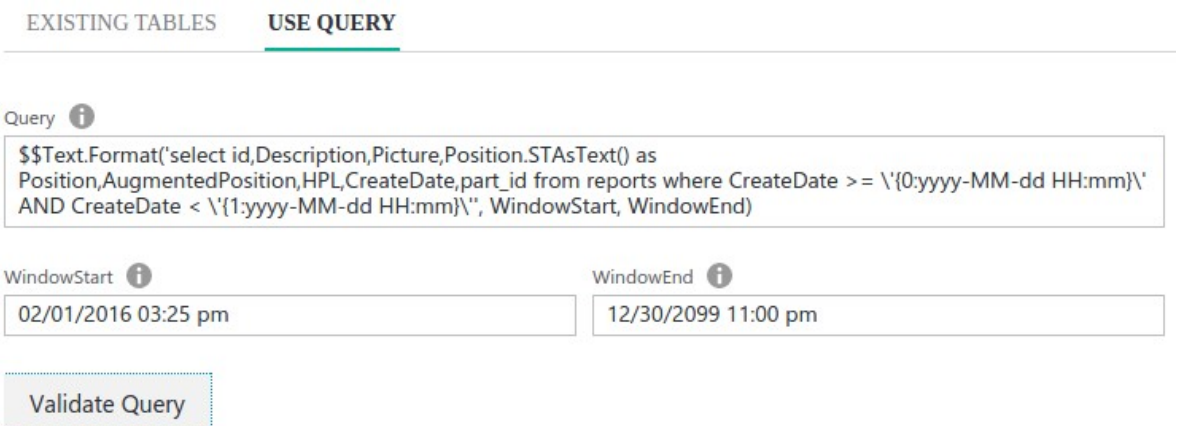


Figura 8.6 Impostazione Query per l'esportazione

Il campo relativo alla query è il seguente:

```
$$Text.Format('select id,
                    Description,
                    Picture,
                    Position.STAsText() as Position,
                    AugmentedPosition,
                    HPL,
                    CreateDate,
                    part_id
              from reports
              where  CreateDate >= \'{0:yyyy-MM-dd HH:mm}\' AND
                     CreateDate < \'{1:yyyy-MM-dd HH:mm}\",
              WindowStart,
              WindowEnd
)
```

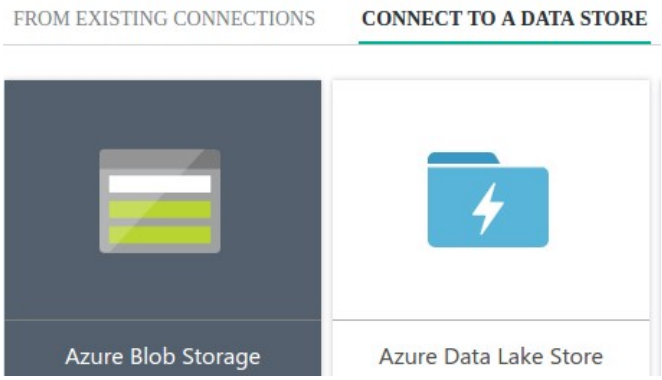


Figura 8.7 Selezione DataStore di Destinazione

Come Data Store di destinazione indichiamo il Blob Storage

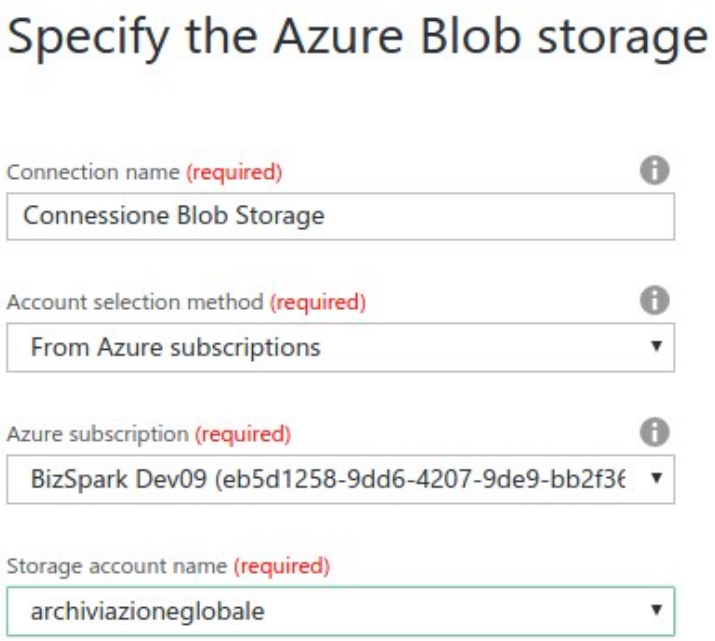


Figura 8.8 Selezione Blob Storage

Specifichiamo i dettagli del Blob Storage di destinazione e cartella e nome file di destinazione

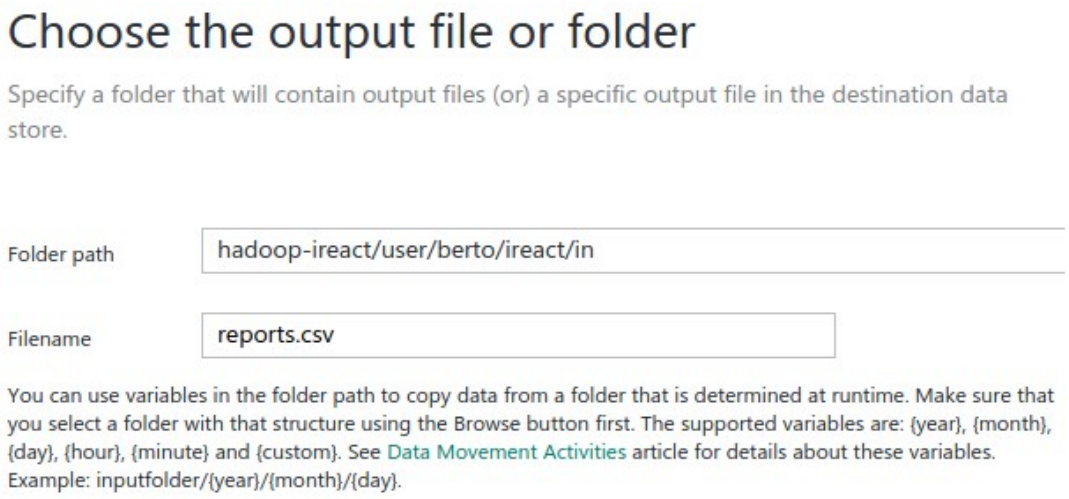


Figura 8.9 Selezione Directory HDFS di Destinazione

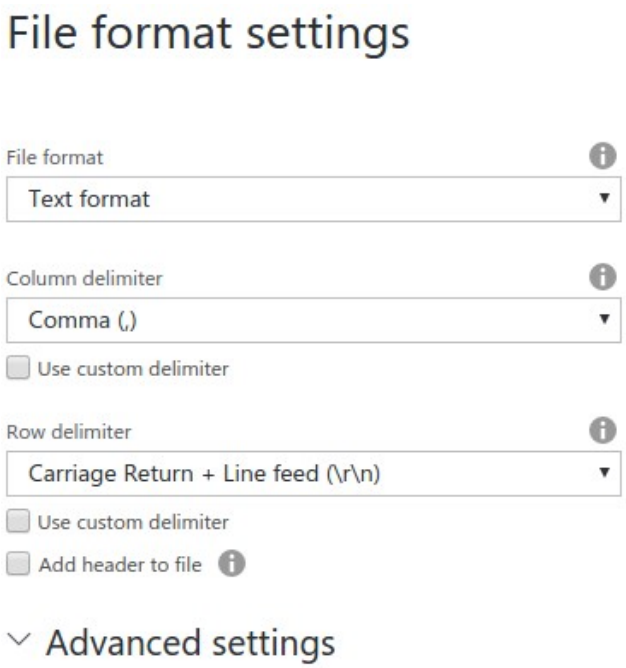


Figura 8.10 Dettagli per l'Esportazione CSV

E' possibile aggiungere dei dettagli sul formato del file da creare (CSV, delimitatore di colonna, ecc.) e delle ottimizzazioni sulle performance

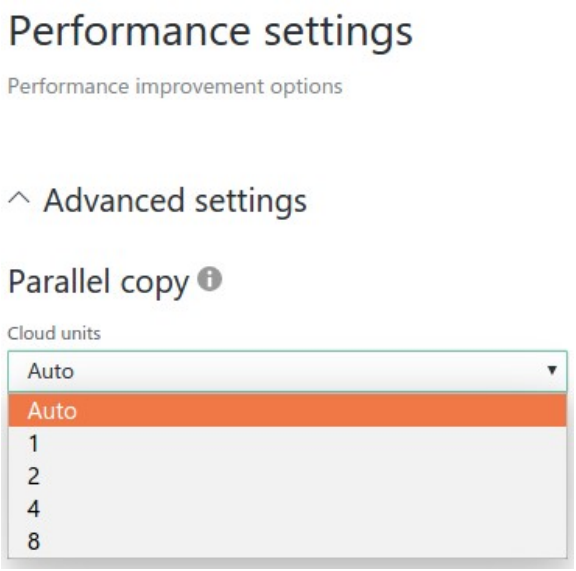


Figura 8.11 Dettagli per l'ottimizzazione dell'esportazione

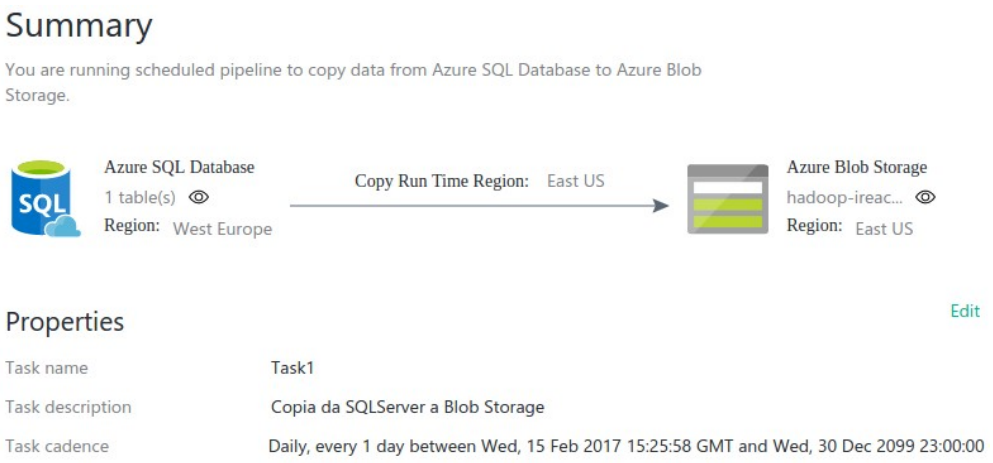


Figura 8.12 Riepilogo Esportazione

L'ultimo passo mostra un riepilogo dell'attività che si sta per eseguire.

Una volta terminata l'attività di esportazione dati, sarà possibile accedere al file testuale prodotto nella cartella determinata; il prossimo passo sarà creare il Cluster HDInsight.

8.2 Creazione Cluster HDInsight

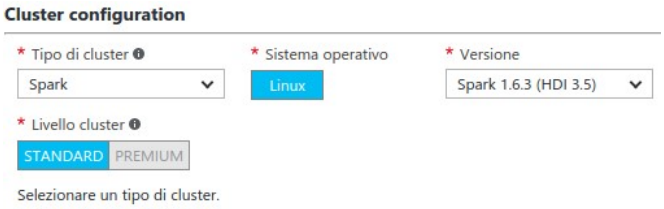


Figura 8.13 Creazione Cluster HDInsight

Nella fase di creazione del Cluster Hadoop, come primo passo è necessario selezionare il tipo di cluster (Spark) e la versione di Spark.

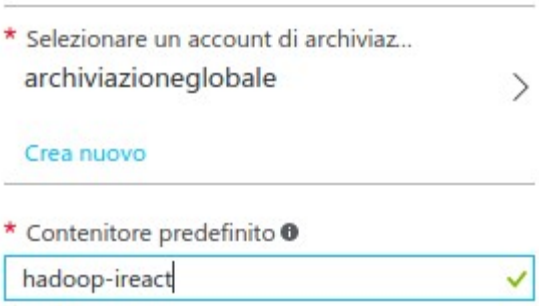


Figura 8.14 Selezione Contenitore Blob Storage

Il passo più importante per poter utilizzare il Blob Storage in maniera efficiente è quando si specificano le impostazioni dello Storage Account: sarà necessario selezionare l'account di archiviazione creato in precedenza di tipo Blob Storage (nella figura si chiama "archiviazioneglobale"), ma soprattutto riutilizzare lo stesso contenitore interno al Blob Storage (che nel nostro esempio si chiama hadoop-ireact), altrimenti verrà creato un contenitore nuovo senza la possibilità di accedere ai dati precedenti.

La stima del tempo impiegato per creare e rendere attivo un Cluster Hadoop è di circa 15 minuti.

Figura 8.15 Dettagli sul Cluster

Nel pannello di controllo del Cluster è presente una stima del consumo orario: ad esempio configurando 6 nodi con 40 cores totali si perviene ad un costo stimato di circa 3.06 €/ora.

Capitolo 9 Tecnologie alternative di Cloud Computing

Amazon: creare RDS, creare EMR, collegarlo a S3, scriptarlo, calcolare tempi partenze e costi, riportare convenienza nell'aumentare le risorse per accelerare la rapidità di risposta.
L'alternativa all'utilizzo del Cloud Computing di Microsoft Azure è quello di Amazon Web Services.

La soluzione che implementa Hadoop su AWS, equivalente alla soluzione Cluster HDInsight di Microsoft Azure, si chiama EMR.

Amazon AWS presenta alcuni aspetti simili al Cloud Computing di Microsoft Azure:

1. In maniera simile a Microsoft Azure, viene suggerito l'utilizzo di uno Storage esterno a basso costo per i dati relativi all'HDFS; questo Storage si chiama S3
2. quando viene cancellato il Cluster EMR, i dati di HDFS rimangono presenti su S3, potendo quindi essere riutilizzati successivamente da una nuovo Cluster EMR

Amazon AWS introduce alcuni importanti novità all'interno della sua infrastruttura di Cloud Computing:

1. Le macchine di tipo T2 possono essere configurate in maniera "scalabile", cioè in fase di partenza della macchina viene assegnata una quantità predefinita di Crediti CPU/ora: nel momento in cui viene richiesta ulteriore CPU, tramite i crediti accumulati si potrà accedere all'intera capacità di elaborazione, per il tempo deciso dai crediti presenti.
2. In maniera molto semplice, tramite uno script, è facile schedulare una cancellazione o un "upgrade" delle caratteristiche hardware di un Cluster Hadoop. Addirittura con Amazon AWS è possibile associare questi eventi in base a determinate condizioni, ad esempio un carico elevato una macchina: questa funzionalità viene chiamata Auto-Scaling, quindi la possibilità di effettuare un upgrade (o un downgrade) in tempo reale delle capacità di calcolo e, soprattutto, in maniera totalmente automatica; questo strumento è nato nelle istanze EC2 (Amazon Elastic Compute Cloud) ma recentemente viene impiegato anche nei Cluster EMR di tipo Hadoop.

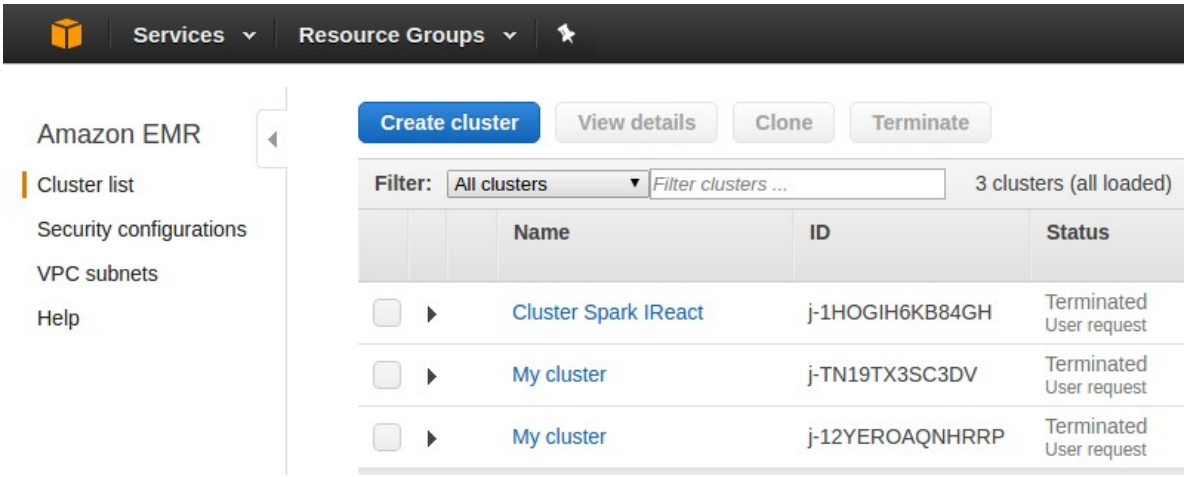


Figura 9.1 Elenco Cluster EMR

Nella figura 9.1 viene mostrato un esempio di schermata che visualizza l'elenco dei Cluster (attivi o eliminati). Da qui è possibile:

1. Creare un Cluster ex-novo
2. Clonare un Cluster esistito e cancellato, per poter effettuare le operazioni di configurazione in maniera più rapida.

9.1 Creazione Cluster Ex novo su Amazon EMR

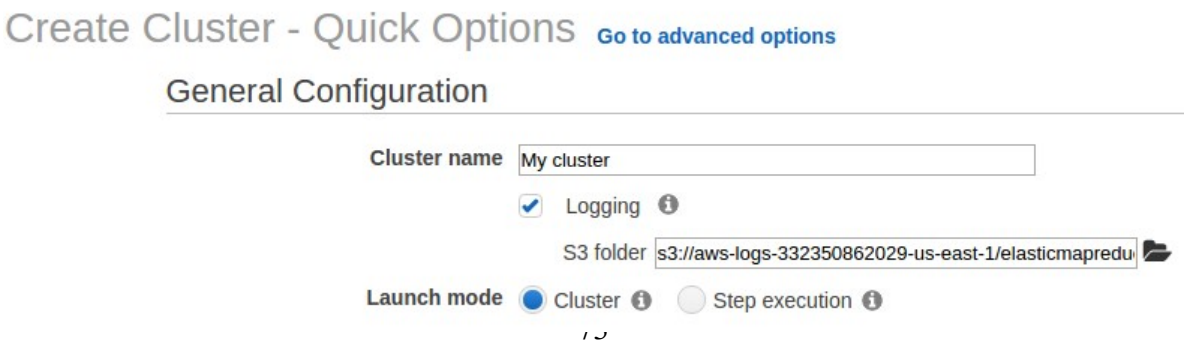


Figura 9.2 Creazione Cluster EMR ex-novo

Dalla prima figura si può notare la prima parte di creazione di un cluster EMR, si può scegliere:

1. Il nome del Cluster
2. Se effettuare il Logging o meno
3. La cartella S3 da utilizzare per accedere con interfaccia HDFS
4. Se, una volta eseguiti gli step specificati più avanti, eliminare il Cluster per risparmiare risorse.

Dalla seconda figura sono evidenti le caratteristiche Hardware e Software del Cluster che si sta per creare:

1. Tipo (Amazon o MapR)
2. Versione EMR
3. Tipo di applicazione installata: per l'applicazione in oggetto è importante selezionare l'opzione "Spark"
4. Nella Configurazione Hardware viene deciso il tipo di istanza e il numero di istanze da creare
5. Infine è presente una voce relativa all'accesso e alla sicurezza

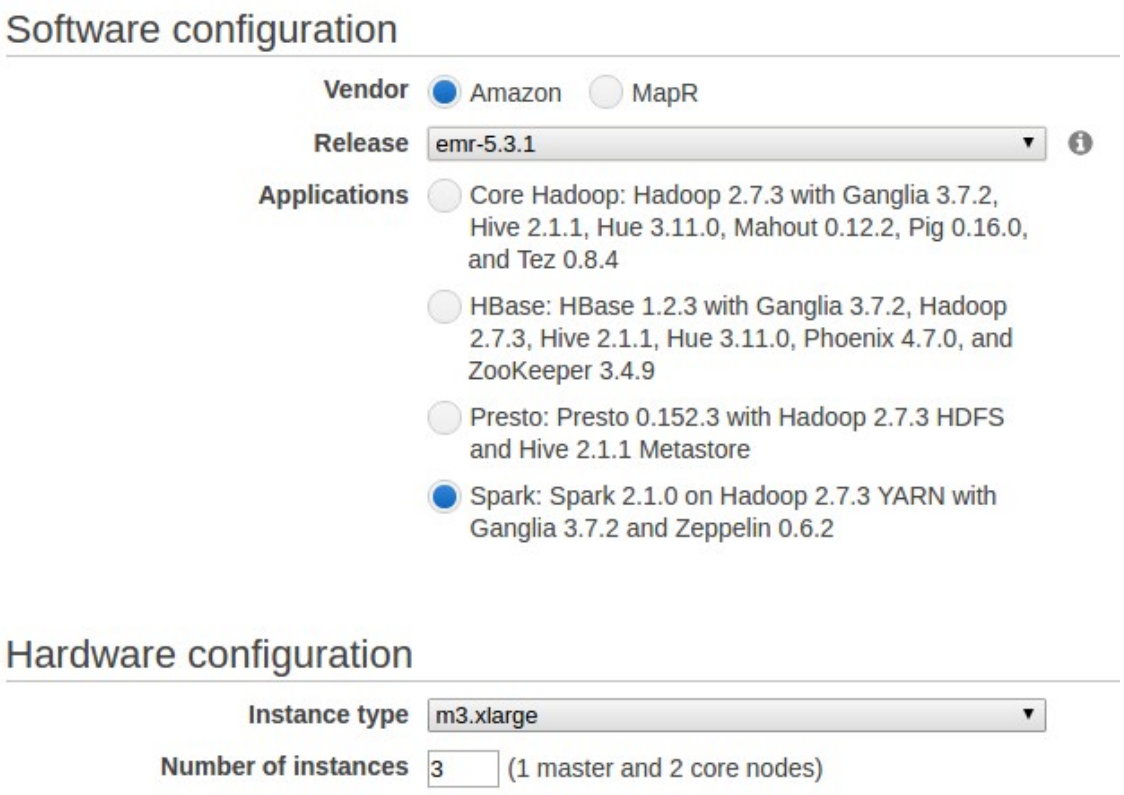


Figura 9.3 Scelta del tipo di Cluster

9.2 Clonazione di un Cluster preesistente su EMR

Se invece si vuole clonare un Cluster già esistito, basta selezionare il cluster e cliccare su Clona.

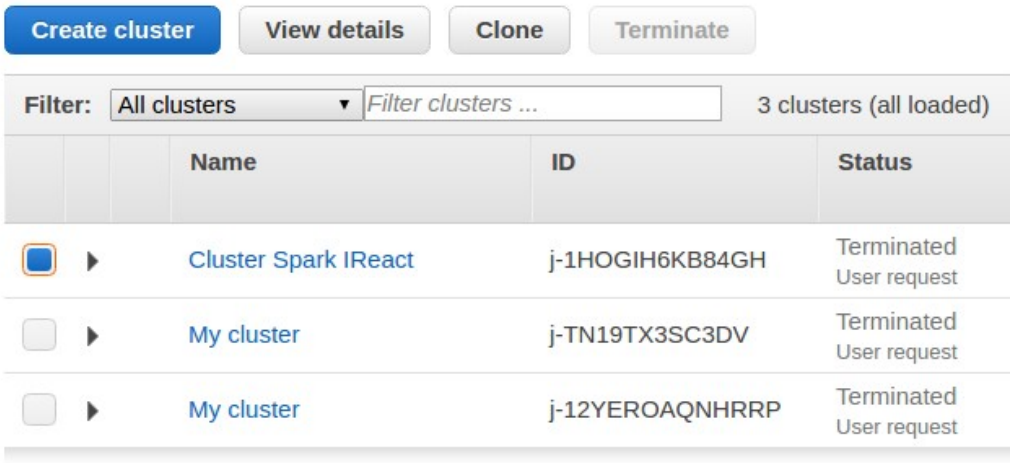


Figura 9.4 Clonazione Cluster

Una nota importante è il bottone "AWS CLI export" che mostra un lungo comando (in questo caso di tipo "create-cluster") che può essere utilizzato in uno script per automatizzare il processo di creazione di un cluster.
Il comando riportato si riferisce ad un Cluster già esistito, per cambiare qualunque attributo o caratteristica è sufficiente modificare il relativo parametro.

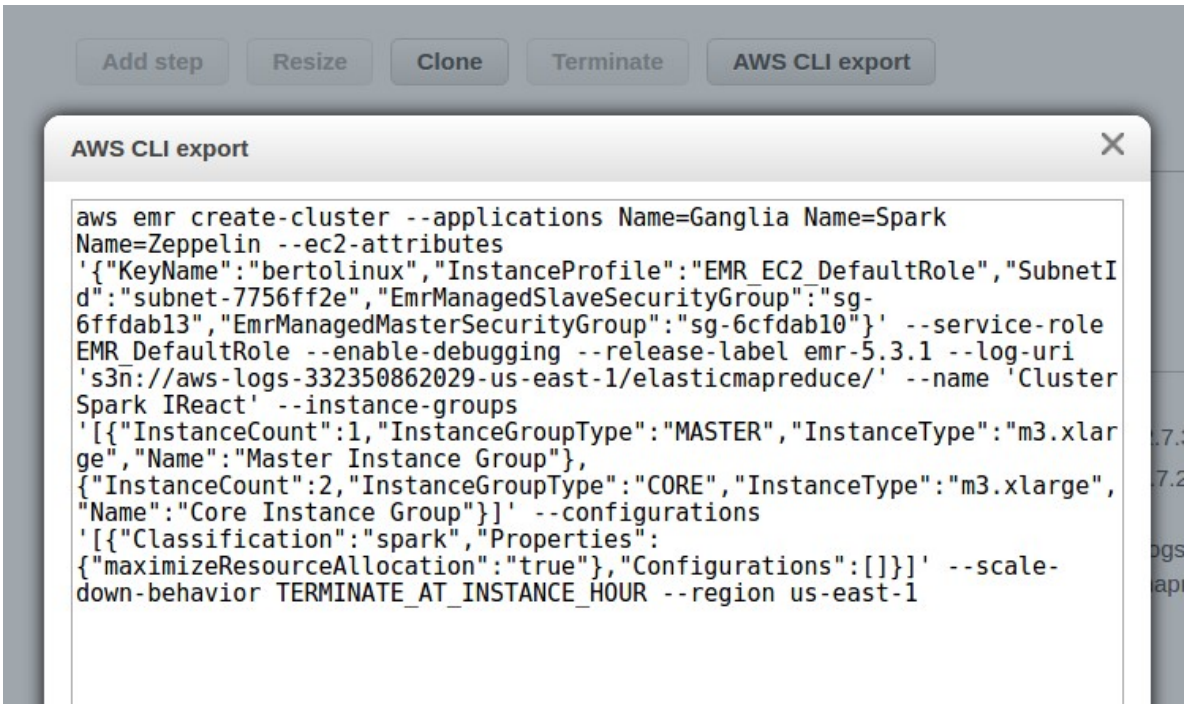


Figura 9.5 Esportazione comando di creazione Cluster

Capitolo 10 Bibliografia e Links

10.1 LIBRI

10.1.1 Java

"Java, La guida completa" - Patrick Naughton, Herbert Schildt - McGrawHill

10.1.2 Hadoop

"Hadoop, The Definitive Guide. Storage and Analysis at Internet Scale" - Tom White - O'Reilly

10.1.3 Spark

"Learning Spark, Lightning-Fast Data Analysis" - Holden Karau, Andy Konwinski, Patrick Wendell & Matei Zaharia - O'Reilly
"Advanced Analytics with Spark, Patterns for Learning from Data at Scale" - Sandy Ryza, Uri Laserson, Sean Owen, Josh Wills – O'Reilly.

10.2 APPLICATIVI SOFTWARE

10.2.1 Apache Hadoop

http://hadoop.apache.org/

10.2.2 Apache Spark

http://spark.apache.org/

10.2.3 Apache Sqoop

http://sqoop.apache.org/

10.2.4 FreeGis

http://freegis.org/

10.2.5 Qgis

https://www.qgis.org/it/site/

10.3 LIBRERIE SOFTWARE

10.3.1 GeoSpark & Babylon

Url: http://geospark.datasyslab.org/
Repository Git: https://github.com/DataSystemsLab/GeoSpark/releases

10.4 CLOUD PROVIDERS

10.4.1 Microsoft Azure

https://azure.microsoft.com/it-it/

10.4.2 Amazon Web Services (AWS)

https://aws.amazon.com/it/

10.5 DATI GEOREFENZIATI

10.5.1 NaturalEarthData

http://www.naturalearthdata.com/

10.5.2 OpenStreetMap Data Extracts

http://download.geofabrik.de/

10.5.3 Database Eventi Sismici

http://www.emsc-csem.org/Earthquake/