FILAUDEAU Eloi, BOURSIER Louis 16/10/20

HeroRank

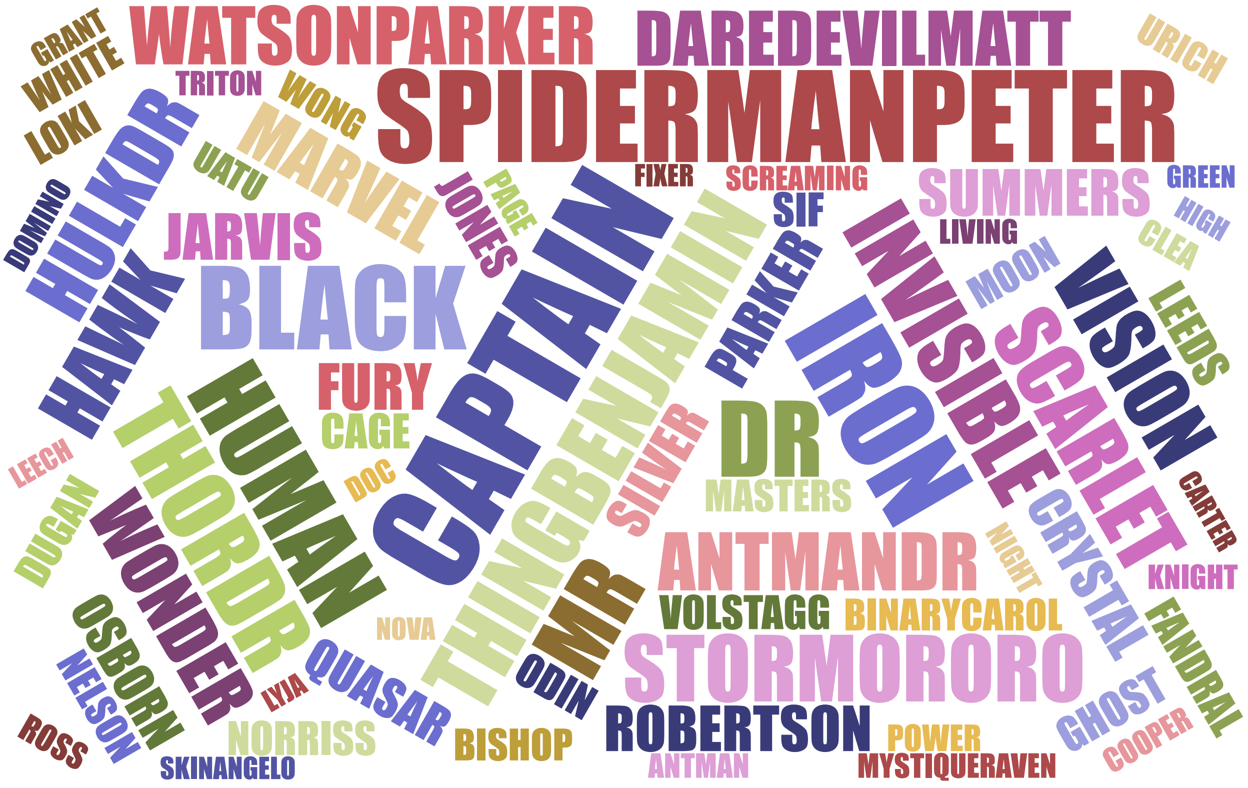


Figure 1: Word cloud generated from the PageRank score of the Marvel heroes

Data origin

Data has been found on the Kaggle website[1], and originates from research work[2]. It is called *The Marvel Universe Social Network, An artificial social network of heroes.* Although PageRank was initially conceived to run on directed graphs (web pages), we will use it on an undirected graph this time. This is because we are using the file called *hero-edge.csv*[3], which contains the network of heroes which appear together in the comics. If one hero meets another one, then so does the other. Hence, edges are coming both way, so we can drop the orientation part.

First, we need to edit the data file to match the format required by our PageRank algorithm.

Original format:

HERO1, HERO2
STEELE, SIMON/WOLFGA","IRON MAN IV/JAMES R.
STEELE, SIMON/WOLFGA","RAVEN, SABBATH II/EL
RAVEN, SABBATH II/EL","FORTUNE, DOMINIC
RAVEN, SABBATH II/EL","ERWIN, CLYTEMNESTRA
RAVEN, SABBATH II/EL","IRON MAN/TONY STARK

Required format:

HeroI	RankI	NeighbourJ	NeighbourK	...
HeroJ	RankJ	NeighbourI	NeighbourK	...

[1] https://www.kaggle.com/csanhueza/the-marvel-universe-social-network
[2] https://arxiv.org/abs/cond-mat/0202174
[3] http://syntagmatic.github.io/exposedata/marvel/

Work required:

- Group by key (hero name)
- Add a rank column at the second place (initialized to 1.0)
- Export file as TSV instead of CSV (because commas are already in use for names)

Starting by the last step, we used OpenRefine to export the CSV file to a TSV file.
For the two other steps, we simply used a Pig script that we ran as a Pig Job on the GCP DataProc interface. Before that, we needed to upload the TSV file to a Bucket on the Google Storage. We would then give the path to this file and to the output file as an argument.

```
r1 = LOAD '$file' USING PigStorage('\t') AS (a:chararray, b:chararray);
r2 = GROUP r1 BY a;
r3 = foreach r2 generate group as a, r1 as b;
r4 = foreach r3 generate a, 1.0, FLATTEN(BagToTuple(b.b));
STORE r4 INTO '$out' USING PigStorage('\t');
```

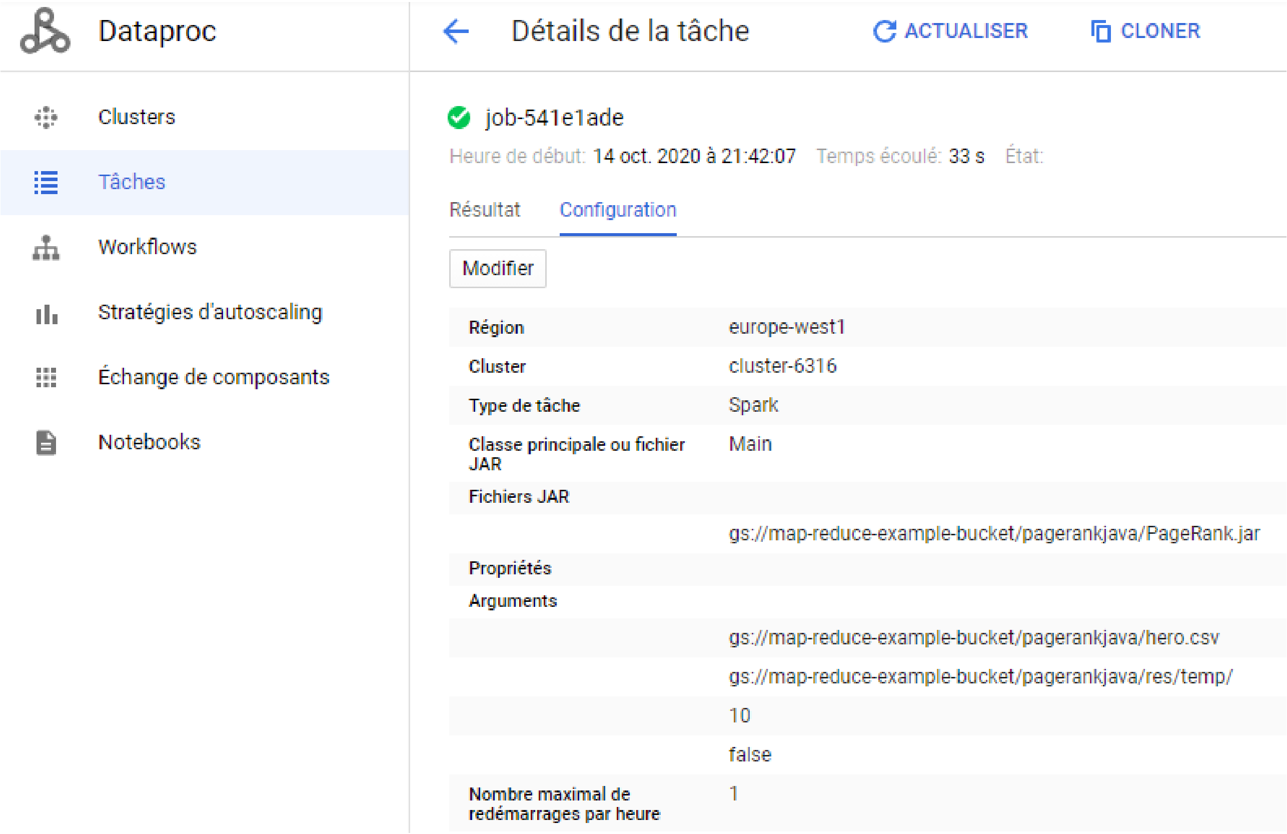
We can now use the output file as the input file for our PageRank program. We wrote the program using the Spark Java API. The source will be available at the end of this document. Here are the key elements of this program:

- The program takes 4 arguments:
  - `<inputPathUri> <outFolderPathUri> <iterations> <debug>`
- First iteration reads data from disk, all the others from RDDs

The algorithm is a simplified version:

- Start each page at rank 1
- On each iteration, have page P contribute to rank(p) / | neighbours(p) |
- Set each page rank to 0.15 + 0.85 * contributions

We compiled and exported the program to a JAR locally, using Maven. We then uploaded this JAR to the Bucket we already set up beforehand for the data preparation, and launched a VM on the same region as the Bucket in the Dataproc drawer of the GCP console. We could finally launch the Spark Job.

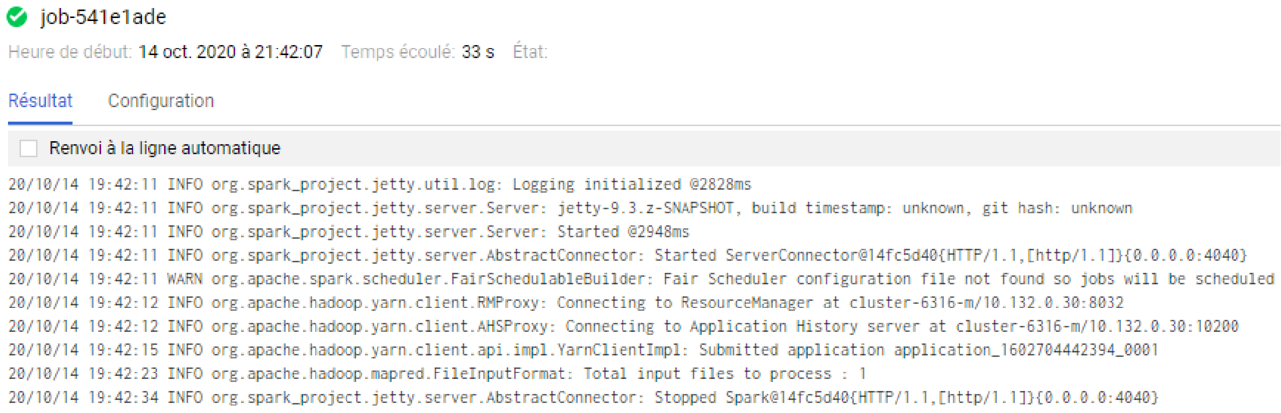


job-541e1ade

Heure de début: 14 oct. 2020 à 21:42:07 Temps écoulé: 33 s État:

Région	europe-west1
Cluster	cluster-6316
Type de tâche	Spark
Classe principale ou fichier JAR	Main
Fichiers JAR	
	gs://map-reduce-example-bucket/pagerankjava/PageRank.jar
Propriétés	
Arguments	
	gs://map-reduce-example-bucket/pagerankjava/hero.csv
	gs://map-reduce-example-bucket/pagerankjava/res/temp/
	10
	false
Nombre maximal de redémarrages par heure	1

Figure 2: Spark Job parameters



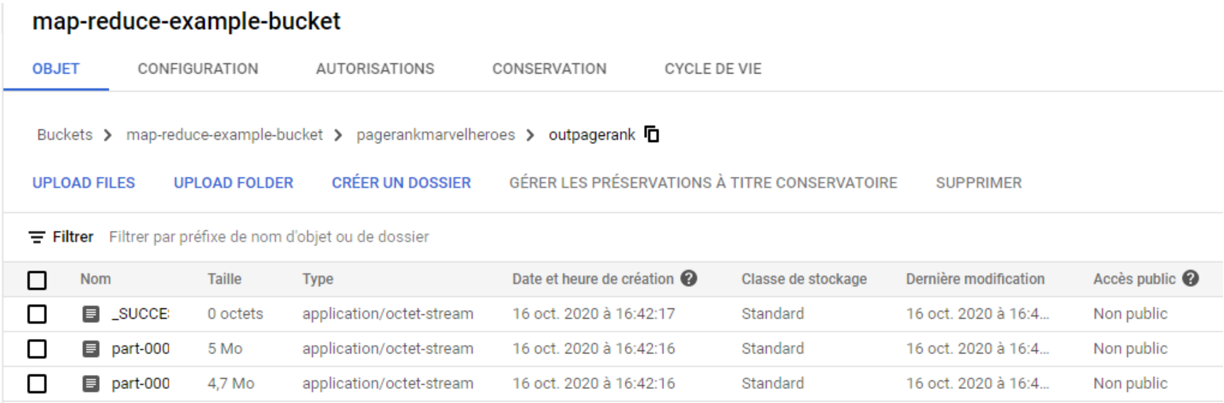
job-541e1ade

Heure de début: 14 oct. 2020 à 21:42:07 Temps écoulé: 33 s État:

□ Renvoi à la ligne automatique

```
20/10/14 19:42:11 INFO org.spark_project.jetty.util.log: Logging initialized @2828ms
20/10/14 19:42:11 INFO org.spark_project.jetty.server.Server: jetty-9.3.z-SNAPSHOT, build timestamp: unknown, git hash: unknown
20/10/14 19:42:11 INFO org.spark_project.jetty.server.Server: Started @2948ms
20/10/14 19:42:11 INFO org.spark_project.jetty.server.AbstractConnector: Started ServerConnector@14fc5d40{HTTP/1.1,[http/1.1]}{0.0.0.0:4040}
20/10/14 19:42:11 WARN org.apache.spark.scheduler.FairSchedulableBuilder: Fair Scheduler configuration file not found so jobs will be scheduled
20/10/14 19:42:12 INFO org.apache.hadoop.yarn.client.RMProxy: Connecting to ResourceManager at cluster-6316-m/10.132.0.30:8032
20/10/14 19:42:12 INFO org.apache.hadoop.yarn.client.AHSProxy: Connecting to Application History server at cluster-6316-m/10.132.0.30:10200
20/10/14 19:42:15 INFO org.apache.hadoop.yarn.client.api.impl.YarnClientImpl: Submitted application application_1602704442394_0001
20/10/14 19:42:23 INFO org.apache.hadoop.mapred.FileInputFormat: Total input files to process : 1
20/10/14 19:42:34 INFO org.spark_project.jetty.server.AbstractConnector: Stopped Spark@14fc5d40{HTTP/1.1,[http/1.1]}{0.0.0.0:4040}
```

Figure 3: Spark Job result

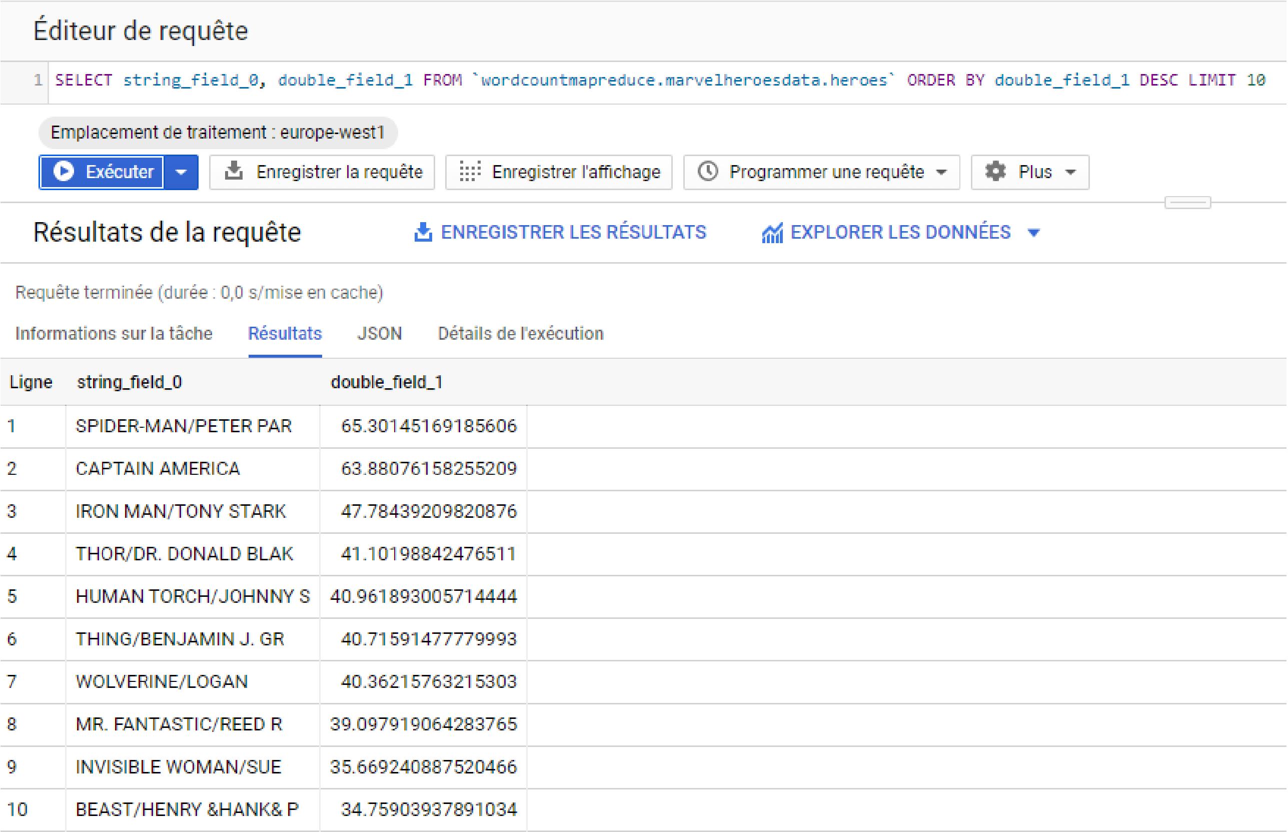

map-reduce-example-bucket
OBJET
CONFIGURATION
AUTORISATIONS
CONSERVATION
CYCLE DE VIE
Buckets > map-reduce-example-bucket > pagerankmarvelheroes > outpagerank
UPLOAD FILES
UPLOAD FOLDER
CRÉER UN DOSSIER
GÉRER LES PRÉSERVATIONS À TITRE CONSERVATOIRE
SUPPRIMER
Filtrer Filtrer par préfixe de nom d'objet ou de dossier

□	Nom	Taille	Type	Date et heure de création	Classe de stockage	Dernière modification	Accès public
□	_SUCCE	0 octets	application/octet-stream	16 oct. 2020 à 16:42:17	Standard	16 oct. 2020 à 16:4...	Non public
□	part-000	5 Mo	application/octet-stream	16 oct. 2020 à 16:42:16	Standard	16 oct. 2020 à 16:4...	Non public
□	part-000	4,7 Mo	application/octet-stream	16 oct. 2020 à 16:42:16	Standard	16 oct. 2020 à 16:4...	Non public

Figure 4: Files produced by the Spark Job

To quickly and easily explore the results, we used Google Big Query. We first had to create a dataset. After that, we created a table automatically by letting Big Query infer the schema from the first file produced by Spark in the GCS bucket. We imported it as a CSV with tab as separators. Finally, we added the second file to the existing table.

Interestingly, our PageRank scores match results found by the study: *"Finally, we have computed the center of the giant component, the character that minimizes the sum of the distances from it to all other nodes in the component. It turns out to be Captain America, who is, on average, at distance 1.70 to every other character."*[4]

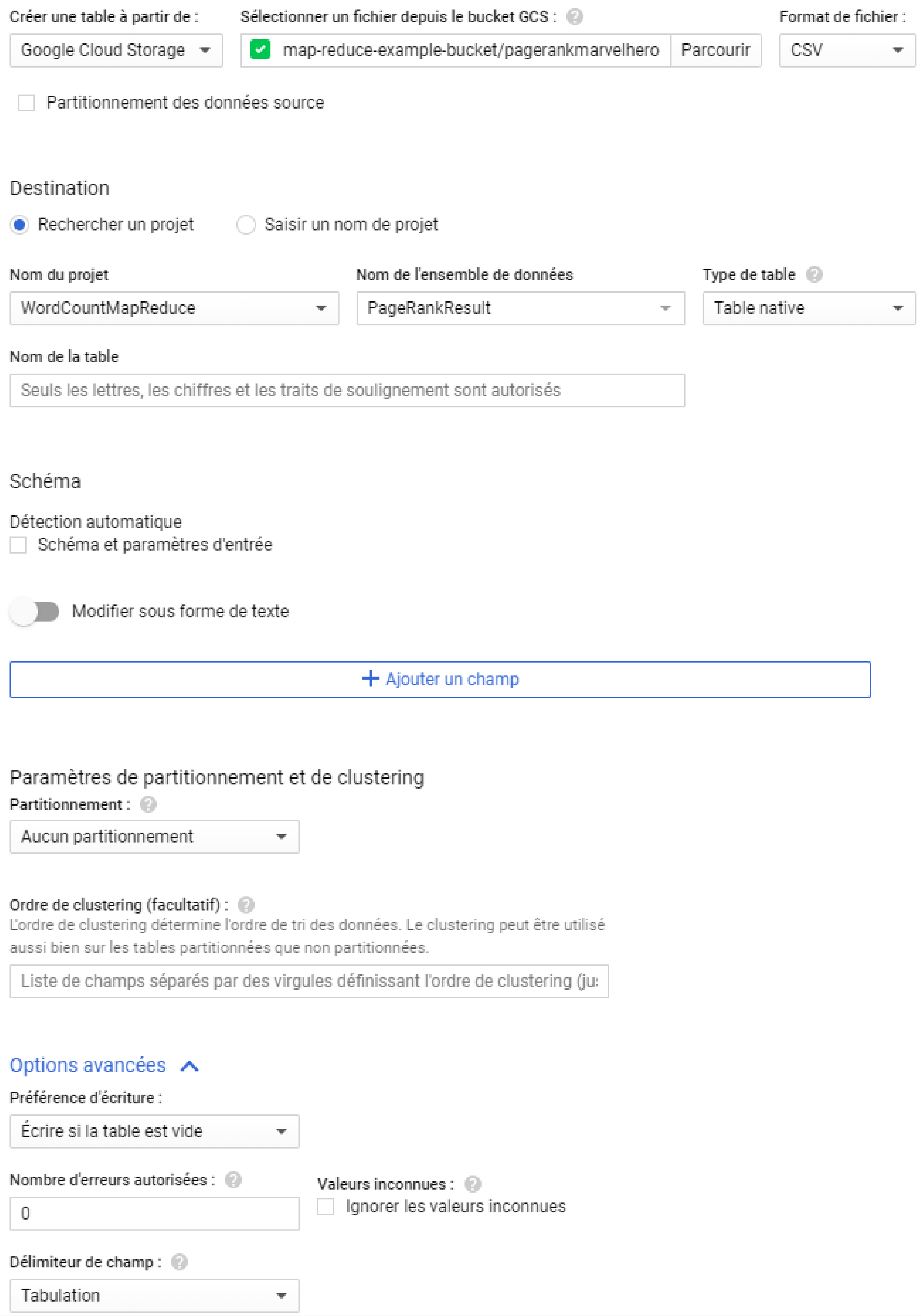


```
SELECT string_field_0, double_field_1 FROM `wordcountmapreduce.marvelheroesdata.heroes` ORDER BY double_field_1 DESC LIMIT 10
```

Ligne	string_field_0	double_field_1
1	SPIDER-MAN/PETER PAR	65.30145169185606
2	CAPTAIN AMERICA	63.88076158255209
3	IRON MAN/TONY STARK	47.78439209820876
4	THOR/DR. DONALD BLAK	41.10198842476511
5	HUMAN TORCH/JOHNNY S	40.961893005714444
6	THING/BENJAMIN J. GR	40.71591477779993
7	WOLVERINE/LOGAN	40.36215763215303
8	MR. FANTASTIC/REED R	39.097919064283765
9	INVISIBLE WOMAN/SUE	35.669240887520466
10	BEAST/HENRY &HANK& P	34.75903937891034

Figure 5: The top 10 heroes by the PageRank score

[4] https://arxiv.org/pdf/cond-mat/0202174.pdf



Créer une table à partir de : Google Cloud Storage

Sélectionner un fichier depuis le bucket GCS : ✓ map-reduce-example-bucket/pagerankmarvelhero Parcourir

Format de fichier : CSV

□ Partitionnement des données source

Destination

● Rechercher un projet ○ Saisir un nom de projet

Nom du projet : WordCountMapReduce

Nom de l'ensemble de données : PageRankResult

Type de table : Table native

Nom de la table : Seuls les lettres, les chiffres et les traits de soulignement sont autorisés

Schéma

Détection automatique

□ Schéma et paramètres d'entrée

Modifier sous forme de texte

Paramètres de partitionnement et de clustering

Partitionnement : Aucun partitionnement

Ordre de clustering (facultatif) :

L'ordre de clustering détermine l'ordre de tri des données. Le clustering peut être utilisé aussi bien sur les tables partitionnées que non partitionnées.

Liste de champs séparés par des virgules définissant l'ordre de clustering (ju:

Options avancées

Préférence d'écriture : Écrire si la table est vide

Nombre d'erreurs autorisées : 0

Valeurs inconnues : □ Ignorer les valeurs inconnues

Délimiteur de champ : Tabulation

```java
import org.apache.spark.api.java.JavaPairRDD;
import org.apache.spark.api.java.JavaSparkContext;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.SparkConf;
import scala.Tuple2;
import java.util.ArrayList;
import java.util.Arrays;

/**
 * Author: Louis Boursier
 * louisboursier@hotmail.fr
 *
 * PageRank Algorithm:
 * 1. Start each page at rank 1
 * 2. on each iteration, have page p contribute to rank(p) / |neighbors(p)| to its neighbors
 * 3. Set each page rank to 0.15 + 0.85 * contributions
 *
 */

public class Main {

    public static final boolean LOCAL_MODE = false;
    public static final int MAX_ITERATIONS = 100;
    public static JavaSparkContext sparkContext; // to avoid JavaSparkContext not serializable error

    public static void main(String[] args) {

        if (args.length != 4) {
            throw new IllegalArgumentException("Exactly 4 arguments are required: " +
                    "<inputPathUri> " +
                    "<outputPathUri> " +
                    "<numberOfIterations> " +
                    "<debugMode>");
        }

        String inputPath = args[0];
        String outputPath = args[1];
        int iterations = Math.min(MAX_ITERATIONS, Integer.valueOf(args[2]));
        boolean debugMode = Boolean.valueOf(args[3]);

        JavaSparkContext sparkContext = null;
        if(LOCAL_MODE) {
            sparkContext = new JavaSparkContext(new SparkConf().setAppName("SparkTestJava").setMaster("local"));
        } else {
            sparkContext = new JavaSparkContext(new SparkConf().setAppName("SparkTestJava"));
        }

        JavaRDD<String> inputLines = null;

        for(int iteration=0 ; iteration<iterations ; iteration++) {

            // Load data from disk the first time, and then from RAM to reduce disk IO latency
            if(iteration == 0) { inputLines = sparkContext.textFile(inputPath); }

            // Saves neighbors for each key, will be used later
            JavaPairRDD<String, String> neighbors = inputLines.mapToPair(line -> {
                String[] lines = line.split("\\t");
                String key = lines[0];
                StringBuilder sb = new StringBuilder();
                String[] neighborsArray = Arrays.copyOfRange(lines, 2, lines.length);
                int i;
                for(i=0 ; i<neighborsArray.length-1 ; i++) { sb.append(neighborsArray[i] + "\t"); }
                if(i<lines.length) { sb.append(neighborsArray[i]); }
                return new Tuple2<>(key, sb.toString());
            });
            // Map
            JavaPairRDD<String, Double> flattenContributions = inputLines.flatMapToPair((String line) -> {
                String[] lines = line.split("\\t");
                String key = lines[0];
                Double rank = Double.valueOf(lines[1]);
                String[] neighborsArray = Arrays.copyOfRange(lines, 2, lines.length);
                ArrayList<Tuple2<String, Double>> contributions = new ArrayList<>();
                for(String neighbor : neighborsArray) {
                    contributions.add(new Tuple2<>(neighbor,(rank/(Double.valueOf(neighborsArray.length)))));
                }
                return contributions.iterator();
            });

            // Reduce
            JavaPairRDD<String, Double> aggregatedContributions =
                    flattenContributions.reduceByKey((aDouble, aDouble2) -> aDouble + aDouble2);

            // Compute rank according to the specified equation
            JavaPairRDD<String, Double> updatedContributions =
                    aggregatedContributions.mapValues(value -> value*0.85+0.15);

            // String cast because join needs to operate on similar data types
            JavaPairRDD<String, String> strUpdatedContributions = updatedContributions.mapValues(value ->
                    String.valueOf(value));

            JavaPairRDD<String, Tuple2<String, String>> join = strUpdatedContributions.join(neighbors);

            // Format it so that it corresponds to the file read at the beginning so that we can iterate over it again
            inputLines = join.map(stringTuple2Tuple2 ->
                    stringTuple2Tuple2._1 + "\t" +
                    stringTuple2Tuple2._2._1 + "\t" +
                    stringTuple2Tuple2._2._2);
        }
        if(debugMode) { inputLines.collect().forEach(s -> System.out.println(s)); }
        if(!LOCAL_MODE) { inputLines.saveAsTextFile(outputPath); }
    }

    // cb1e050d9f48e5cb734270d96678f089
}
```