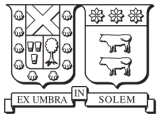

UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA

Trabajo Práctico 2

INF356 - 2025-1 - 200

Computación Distribuida para Big Data

9 de junio de 2025 - v1.0

Miguel Soto | 201973623-K
miguel.sotod@sansano.usm.cl

Índice

Índice de tablas

Índice de figuras

Índice de fragmentos de codigo

1. Map Reduce

Esta entrega consistia en modificar el codigo fuente de Hadoop para agregar funcionalidad extra, en donde habia que implementar metodos a partir de la funcion WordCount.

Se pedia que crear una funcion que permitiese contar palabras previamente filtradas a partir de reglas indicadas en el informe.

Luego, habia que crear otra funcion que permitiese extraer columnas de la salida de WordCount a partir del indice de la columna.

Por motivos de eficiencia y por simplicidad, opte por parchar el codigo binario de Hadoop con las clases nuevas creadas a partir de WordCount en las funciones de ejemplo originales, de esta forma pude invocarlas facilmente y no tener que compilar el codigo fuente cada vez que hiciera un cambio en el codigo.

Grabe un video explicativo mostrando el proceso en ejecucion junto a las salidas de ambos programas, ya que en el practico inicial fue recurrente el tema de que no dejar evidencia clara de los puntos que iba desarrollando. Aqui deje el enlace al video en YouTube.

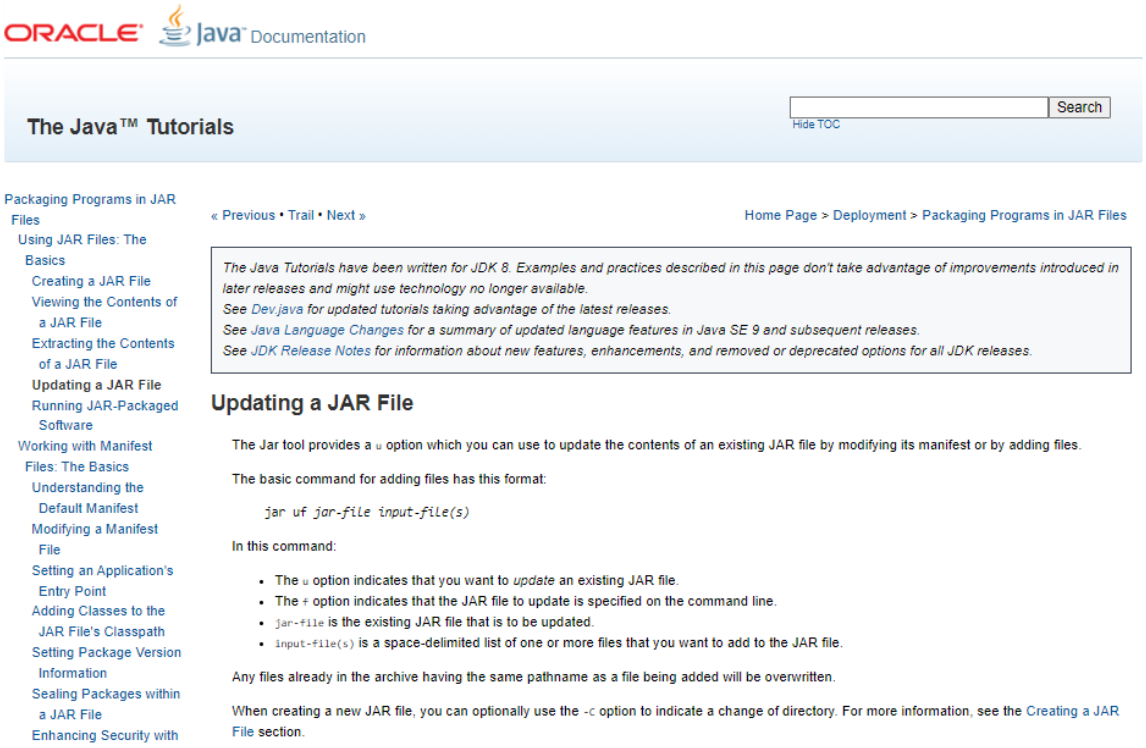

Figura 1: Documentacion oficial de Oracle sobre como parchar un JAR

1.1. Implementación extendida de wordcounter

Para desarrollar esta nueva funcionalidad me base en la implementacion de la clase original de WordCount, la cual obtuve desde el repositorio publico de Apache Hadoop en GitHub (enlace), en donde dicha funcion se encuentra en la carpeta de ejemplos.

La forma en la que luego parche el codigo, fue aprovechandome de los .jar que contiene el binario de Hadoop, estos ultimos son esencialmente archivos comprimidos que contienen los archivos .class de todas las clases .java de la fuente. Mas aun, el comando jar tiene la funcionalidad de actualizar un .jar con clases nuevas o con modificaciones de estas. Esto se puede lograr de la siguiente manera:

A continuacion dejare el codigo original de WordCount y el codigo de ExtendedWordCount, hecho a partir del anterior.

```java
/**
 * Licensed to the Apache Software Foundation (ASF) under one
 * or more contributor license agreements.  See the NOTICE file
 * distributed with this work for additional information
 * regarding copyright ownership.  The ASF licenses this file
 * to you under the Apache License, Version 2.0 (the
 * "License"); you may not use this file except in compliance
 * with the License.  You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
package org.apache.hadoop.examples;

import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
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.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;

public class WordCount {

 public static class TokenizerMapper
      extends Mapper<Object, Text, Text, IntWritable>{

   private final static IntWritable one = new IntWritable(1);
   private Text word = new Text();

   public void map(Object key, Text value, Context context
                   ) throws IOException, InterruptedException {
     StringTokenizer itr = new StringTokenizer(value.toString());
     while (itr.hasMoreTokens()) {
       word.set(itr.nextToken());
       context.write(word, one);
     }
   }
 }

 public static class IntSumReducer
      extends Reducer<Text,IntWritable,Text,IntWritable> {
   private IntWritable result = new IntWritable();

   public void reduce(Text key, Iterable<IntWritable> values,
                      Context context
                      ) throws IOException, InterruptedException {
     int sum = 0;
     for (IntWritable val : values) {
       sum += val.get();
     }
     result.set(sum);
     context.write(key, result);
   }
 }

 public static void main(String[] args) throws Exception {
   Configuration conf = new Configuration();
   String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
   if (otherArgs.length < 2) {
     System.err.println("Usage:_wordcount_<in>_[<in>...]_<out>");
     System.exit(2);
   }
   Job job = Job.getInstance(conf, "word_count");
   job.setJarByClass(WordCount.class);
   job.setMapperClass(TokenizerMapper.class);
   job.setCombinerClass(IntSumReducer.class);
   job.setReducerClass(IntSumReducer.class);
   job.setOutputKeyClass(Text.class);
   job.setOutputValueClass(IntWritable.class);
   for (int i = 0; i < otherArgs.length - 1; ++i) {
     FileInputFormat.addInputPath(job, new Path(otherArgs[i]));
   }
   FileOutputFormat.setOutputPath(job,
     new Path(otherArgs[otherArgs.length - 1]));
   System.exit(job.waitForCompletion(true) ? 0 : 1);
 }
}
```

Código 1: Codigo original de WordCount

```java
package org.apache.hadoop.examples;

import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
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.Mapper;
import org.apache.hadoop.mapreduce.Reducer;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class ExtendedWordCount {

  public static class TokenizerMapper
       extends Mapper<Object, Text, Text, IntWritable> {

    private final static IntWritable one = new IntWritable(1);
    private final Text word = new Text();

    @Override
    public void map(Object key, Text value, Context context)
        throws IOException, InterruptedException {

      String line = value.toString();
      StringTokenizer tokenizer = new StringTokenizer(line);

      while (tokenizer.hasMoreTokens()) {
        String token = tokenizer.nextToken();

        // 1. Convertir a minúsculas
        token = token.toLowerCase();

        // 2. Eliminar puntuaciones al inicio y al final (regex)
        token = token.replaceAll("^[^\\p{L}\\p{Nd}]+|[^\\p{L}\\p{Nd}]+$", "");

        // 3. Descartar si es solo puntuación o está vacío
        if (token.matches("^[^\\p{L}\\p{Nd}]+$") || token.isEmpty()) {
          continue;
        }

        word.set(token);
        context.write(word, one);
      }
    }
  }

  public static class IntSumReducer
       extends Reducer<Text, IntWritable, Text, IntWritable> {

    private final IntWritable result = new IntWritable();

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

      int sum = 0;
      for (IntWritable val : values) {
        sum += val.get();
      }

      result.set(sum);
      context.write(key, result);
    }
  }

  public static void main(String[] args) throws Exception {
    if (args.length != 2) {
      System.err.println("Uso: ExtendedWordCount <input_path> <output_path>");
      System.exit(-1);
    }

    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "extended word count");

    job.setJarByClass(ExtendedWordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);

    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);

    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```


Código 2: Codigo de ExtendedWordCount

1.2. Implementación de selector de columna

De la misma manera que el punto anterior, me base en WordCount para crear el codigo para la clase SelectColumnExtractor, el cual permite sustraer una columna de la salida de WordCount en base al indice/numero de columna. Esta funcion en particular ejecuta WordCount si lo filtra al terminar el proceso de Map-Reduce.

```java
package org.apache.hadoop.examples;

import java.io.IOException;

import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

public class SelectColumnExtractor extends Configured implements Tool {

    public static class ColumnMapper extends Mapper<LongWritable, Text, NullWritable, Text> {
        private int selectColumn = -1;

        @Override
        protected void setup(Context context) {
            Configuration conf = context.getConfiguration();
            selectColumn = conf.getInt("select.column", -1);
        }

        @Override
        protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
            String line = value.toString();
            String[] rawColumns = line.split(",(?=\")"); // espacio antes de comillas dobles
            int totalColumns = rawColumns.length;

            if (selectColumn >= totalColumns || selectColumn < 0) {
                context.write(NullWritable.get(), new Text(line));
                return;
            }

            String column = rawColumns[selectColumn].trim();
            if (column.startsWith("\"") && column.endsWith("\"") && column.length() >= 2) {
                column = column.substring(1, column.length() - 1);
            }

            context.write(NullWritable.get(), new Text(column));
        }
    }

    @Override
    public int run(String[] args) throws Exception {
        if (args.length != 3) {
            System.err.println("Usage: SelectColumnExtractor <input> <output> <columnIndex>");
            return -1;
        }

        Configuration conf = getConf();
        conf.setInt("select.column", Integer.parseInt(args[2]));

        Job job = Job.getInstance(conf, "Select Column Extractor");
        job.setJarByClass(SelectColumnExtractor.class);

        job.setMapperClass(ColumnMapper.class);
        job.setNumReduceTasks(0); // solo mapper

        job.setOutputKeyClass(NullWritable.class);
        job.setOutputValueClass(Text.class);

        FileInputFormat.setInputPaths(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        return job.waitForCompletion(true) ? 0 : 1;
    }

    public static void main(String[] args) throws Exception {
        int exitCode = ToolRunner.run(new SelectColumnExtractor(), args);
        System.exit(exitCode);
    }
}
```

Código 3: Codigo de SelectColumnExtractor

1.3. Tiempo de ejecución

A continuación se muestran las medidas obtenidas utilizando el comando time en ExtendedWordCount:

Archivo	real (s)	user (s)	sys (s)
sw-script-e04.txt	0m32.822s	0m7.653s	0m0.388s

Tabla 1: Tiempos de ejecución del WordCount extendido

Luego se muestran las medidas obtenidas utilizando el comando time en SelectColumnExtractor:

Archivo	real (s)	user (s)	sys (s)
sw-script-e04.txt	0m17.656s	0m7.137s	0m0.350s

Tabla 2: Tiempos de ejecución del SelectColumnExtractor

Comparando ambos resultados, puedo concluir que:

- El algoritmo de ExtendedWordCount es más intensivo en CPU, debido al preprocesamiento de texto, pero se mantiene eficiente gracias a la simplicidad del flujo MapReduce.
- El SelectColumnExtractor es más ligero y orientado a E/S, siendo más rápido en archivos simples, pero ligeramente mas costoso si las líneas contienen muchas columnas.
- Ambas soluciones escalan correctamente, pero el SelectColumnExtractor presenta tiempos de respuesta mas bajos en tareas estructuradas y simples.

Los resultados se pueden ver en tiempo real en el video indicado al inicio del practico (enlace).

2. Spark

A continuacion se describe detalladamente el desarrollo de la tarea práctica avanzada de Big Data, basada en el uso de `Flintrock` para la configuración de un clúster Spark en AWS. A diferencia del informe original, este documento se enfoca en detallar todo el proceso personal, incluyendo errores cometidos, correcciones, decisiones de diseño, observaciones y comentarios adicionales que pueden ser útiles para otros estudiantes que enfrenten una tarea similar. Se proporciona ademas un video explicativo mostrando evidencia de el levantamiento de el informe y otros detalles adicionales (enlace).

Para comenzar, se utilizó el mismo clúster que se trabajó en la entrega inicial. Se eliminó manualmente todos los nodos worker, manteniendo solamente el nodo master. Este paso fue esencial para asegurar una base limpia antes de implementar una nueva configuración.

Se clonaron los siguientes repositorios proporcionados por el profesor:

- `https://github.com/ptoledo-teaching/training-bigdata-002`
- `https://github.com/nchammas/flintrock`

Estos repositorios contienen scripts clave para la creación del clúster. Se realizó una navegación rápida con `tree` y `vim` para entender su estructura y funcionalidades.

El proceso de instalación incluyó los siguientes pasos:

1. Instalación de `AWS CLI`.
2. Instalación de `pipx`.
3. Instalación de `Flintrock` mediante `pipx`.

Durante este proceso, se descubrió que el script contenía una instrucción para crear un rol IAM para S3, pero el profesor indicó que esto no era necesario. Por ende, se omitió este paso aunque inicialmente se intentó (sin éxito, debido a permisos insuficientes).

Se detectó un bug en la versión actual de Flintrock relacionado con incompatibilidades de Spark. Para solucionarlo, se aplicó un parche manualmente en el código fuente de Flintrock instalado. El procedimiento incluyó:

- Edición directa de los archivos indicados por el profesor.
- Verificación y corrección de problemas de indentación (uso de tabs vs. espacios).
- Uso de expresiones regulares para reemplazo masivo.

Nota: Se olvidó hacer un respaldo del archivo original antes de modificar, lo que pudo haber provocado una reinstalación completa en caso de error.

Se creó una nueva clave PEM con permisos adecuados y se renombró según el formato requerido por Flintrock. Inicialmente se cometió un error en los permisos, generando un `warning`. Posteriormente se corrigió con:

```
chmod 400 clave.pem
```

Esto permitió una conexión exitosa al nodo master.

2.1. Escalamiento vertical y horizontal

Se procedió a levantar múltiples configuraciones de clúster usando archivos `.yaml` modificados. Se observó que configuraciones con más de 8 nodos fallaban debido a restricciones de la cuenta de estudiante. Por lo tanto, se trabajó principalmente con configuraciones de hasta 6 nodos `t3.large`.

- Configuración con 10 `t3.medium` no logró levantar el clúster.
- Ejecute el script `test-000.sh` en lugar de `test-001.sh`, generando inconsistencias en los benchmarks.

Tiempos Registrados

Configuración	Test Ejecutado	Tiempo Aproximado
4 `t3.large`	test-000.sh	24 (real)
4 `t3.large`	test-000.sh	24 (real)
4 `t3.small`	test-000.sh	25 (real)
6 `t3.micro`	test-000.sh	25 (real)
6 `t3.small`	test-000.sh	26 (real)
10 `t3.medium`	test-000.sh	Falló (fallaba en launch.sh)

Nota: Los tiempos fueron medidos usando el comando `time bash test00.sh`, y registrados manualmente en un bloc de notas para su análisis posterior.

3. Procesamiento de datos

3.1. Extracción, transformación y carga

Desarrolle un programa basado en el test-001 que permita la extracción, transformación y carga (extraction, transformation and load, normalmente denominado ETL) del dataset de la asignatura. El set total de datos de la asignatura corresponde al archivo de 7.2[GB] disponible en `s3://utfsm-inf356-datasets/vlt_observations/vlt_observations.csv`. Este dataset ha sido dividido en 20 segmentos con nombre `s3://utfsm-inf356-datasets/vlt_observations/vlt_observations_XXX.csv` para poder disponer de un acceso fraccionado a los datos.

El proceso ETL debe procesar el dataset entregado y generar un nuevo dataset para procesamiento posterior, el que debe considerar las siguientes columnas:

- Right ascension - grados
- Right ascension - minutos
- Right ascension - segundos
- Declination - grados
- Declination - minutos
- Declination - segundos
- Instrument
- Exposition time
- Template start (en tiempo unix)

El dataset procesado sólo debe tener los registros que corresponden a las observaciones con categoría SCIENCE y tipo de observación OBJECT (referencia `https://archive.eso.org/eso/eso_archive_help.html`).

El resultado del dataset debe ser guardado en formato parquet en la raíz de su bucket de desarrollo bajo el nombre **vlt_observations_etl.parquet**. Se solicitará incluir el código bajo el nombre code-005.py en su entrega. Describa el procedimiento y muestre alguna métrica relativa a la ejecución y/o resultado de su procedimiento.

```python
from pyspark.sql import SparkSession

# Create a SparkSession
spark = SparkSession.builder.getOrCreate()

# Stop Spark session
spark.stop()
```

Código 4: Código utilizado para el procedimiento de ETL

3.2. Coordenadas galácticas

Desarrolle un programa que lea el archivo generado por el proceso ETL y genere un nuevo archivo parquet en el que se han transformado las coordenadas RA-DEC ecuatoriales de las observaciones a coordenadas galácticas. El nuevo archivo debe estar en la raíz de su bucket de desarrollo y debe ser llamado **vlt_observations_gc.parquet**. El archivo debe tener las columnas:

- Galactic right ascension (float en grados)
- Galactic declination (float en grados)
- Instrument
- Exposition time
- Template start (en tiempo unix)

Se solicitará incluir el código bajo el nombre code-006.py en su entrega. Describa el procedimiento y muestre alguna métrica relativa a la ejecución y/o resultado de su procedimiento.

```python
from pyspark.sql import SparkSession

# Create a SparkSession
spark = SparkSession.builder.getOrCreate()

# Stop Spark session
spark.stop()
```

Código 5: Código utilizado para el cálculo de coordenadas galácticas

3.3. Segmentación temporal

Segmente el archivo de coordenadas galácticas en tantos archivos como sea necesario de modo de contar con una agrupación por año y por semana del año. Se considera la primera semana del año la semana del primer lunes de un año. Los primeros días del año pertenecen al año anterior si ocurren antes del primer lunes del año.

Los datos deben ser guardados en formato parquet en una carpeta llamada **partition** en la raiz de su bucket de desarrollo. Dentro de partition las carpetas se separarán por año. Dentro de la carpeta de un determinado año deberá haber un archivo denominado **vlt_observations_XXXX.parquet** con todos los datos del año, y una carpeta denominada weeks, que dentro debe tener una serie de archivos denominados **vlt_observations_XXXX_YY.parquet**, donde XXXX corresponde al año y YY al número de semana comenzando en 0.

Se solicitará incluir el código bajo el nombre code-007.py en su entrega. Describa el procedimiento y muestre alguna métrica relativa a la ejecución y/o resultado de su procedimiento.

```python
from pyspark.sql import SparkSession

# Create a SparkSession
spark = SparkSession.builder.getOrCreate()

# Stop Spark session
spark.stop()
```

Código 6: Código utilizado para segmentación temporal de datos

3.4. Conteo de observaciones

Desarrolle un programa que reciba un archivo parquet con el formato previamente utilizado para las coordenadas galácticas y que genere un conteo de las observaciones para cada segmento de 10 grados horizontal y vertical del cielo, segmentado por cada instrumento. El archivo de salida debe tener el mismo nombre que el archivo de entrada pero con un .count antes de la extensión .parquet. El conteo tiene que tener como coordenada de referencia el centro de la región angular. El tiempo de exposición debe ser reemplazado por el tiempo total de observaciones que han sido considerados en la cuenta. Se debe descartar la columna con el tiempo de inicio del template.

Se solicitará incluir el código bajo el nombre code-008.py en su entrega. Describa el procedimiento y muestre alguna métrica relativa a la ejecución y/o resultado de su procedimiento.

```python
from pyspark.sql import SparkSession

# Create a SparkSession
spark = SparkSession.builder.getOrCreate()

# Stop Spark session
spark.stop()
```

Código 7: Código utilizado para segmentación temporal de datos