

UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA

Trabajo Práctico 2

INF356 - 2025-1 - 200

Computación Distribuida para Big Data
23 de junio de 2025 - v1.0

Índice

1. Map Reduce	3
1.1. Implementación extendida de wordcounter	4
1.2. Implementación de selector de columna	7
1.3. Tiempo de ejecución	9
2. Spark	10
2.1. Escalamiento vertical y horizontal	11
3. Procesamiento de datos	12
3.1. Extracción, transformación y carga	12
3.2. Coordenadas galácticas	18
3.3. Segmentación temporal	21
3.4. Conteo de observaciones	24

Índice de tablas

1. Tiempos de ejecución del WordCount extendido	9
2. Tiempos de ejecución del SelectColumnExtractor	9

Índice de figuras

1. Documentacion oficial de Oracle sobre como parchar un JAR	4
2. Neofetch en el nodo master del cluster de Spark	13
3. Spark procesando los datos de esta sección con el código al lado	13
4. Dataframe inicial y resultante tras procesar los datos	13

5.	Dataframe resultante tras procesar los datos	14
6.	Spark procesando los datos mientras explico el código	18
7.	Dataframe resultante tras procesar los datos	18
8.	Pantallazo del Bucket S3 resultante tras procesar los datos	21
9.	Dataframe resultante tras procesar los datos	24

Índice de fragmentos de código

1.	Código original de WordCount	5
2.	Código de ExtendedWordCount	6
3.	Código de SelectColumnExtractor	8
4.	Código utilizado para el procedimiento de ETL	15
5.	Código utilizado para el procedimiento de ETL	16
6.	Código utilizado para el procedimiento de ETL	17
7.	Código utilizado para el cálculo de coordenadas galácticas	19
8.	Código utilizado para el cálculo de coordenadas galácticas	20
9.	Código utilizado para segmentación temporal de datos	22
10.	Código utilizado para segmentación temporal de datos	23
11.	Código utilizado para segmentación temporal de datos	25

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.

The screenshot shows a section of the Oracle Java Documentation titled "Updating a JAR File". On the left, there's a sidebar with a tree view of topics like "Packaging Programs in JAR Files", "Using JAR Files: The Basics", and "Creating a JAR File". The main content area has a header "The Java™ Tutorials" and a search bar. Below the header, there's a note about the page being written for JDK 8 and mentioning Dev.java for updated tutorials. The main text describes the "u" option of the jar tool for updating JAR files, providing a command example and a list of bullet points explaining its parameters. A note at the bottom says that files already in the archive will be overwritten.

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.

```

1 /**
2 * Licensed to the Apache Software Foundation (ASF) under one
3 * or more contributor license agreements. See the NOTICE file
4 * distributed with this work for additional information
5 * regarding copyright ownership. The ASF licenses this file
6 * to you under the Apache License, Version 2.0 (the
7 * "License"); you may not use this file except in compliance
8 * with the License. You may obtain a copy of the License at
9 *
10 * http://www.apache.org/licenses/LICENSE-2.0
11 *
12 * Unless required by applicable law or agreed to in writing, software
13 * distributed under the License is distributed on an "AS IS" BASIS,
14 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
15 * See the License for the specific language governing permissions and
16 * limitations under the License.
17 */
18 package org.apache.hadoop.examples;
19
20 import java.io.IOException;
21 import java.util.StringTokenizer;
22
23 import org.apache.hadoop.conf.Configuration;
24 import org.apache.hadoop.fs.Path;
25 import org.apache.hadoop.io.IntWritable;
26 import org.apache.hadoop.io.Text;
27 import org.apache.hadoop.mapreduce.Job;
28 import org.apache.hadoop.mapreduce.Mapper;
29 import org.apache.hadoop.mapreduce.Reducer;
30 import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
31 import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
32 import org.apache.hadoop.util.GenericOptionsParser;
33
34 public class WordCount {
35
36     public static class TokenizerMapper
37         extends Mapper<Object, Text, Text, IntWritable>{
38
39         private final static IntWritable one = new IntWritable(1);
40         private Text word = new Text();
41
42         public void map(Object key, Text value, Context context
43                         ) throws IOException, InterruptedException {
44             StringTokenizer itr = new StringTokenizer(value.toString());
45             while (itr.hasMoreTokens()) {
46                 word.set(itr.nextToken());
47                 context.write(word, one);
48             }
49         }
50     }
51
52     public static class IntSumReducer
53         extends Reducer<Text,IntWritable,Text,IntWritable> {
54         private IntWritable result = new IntWritable();
55
56         public void reduce(Text key, Iterable<IntWritable> values,
57                           Context context
58                           ) throws IOException, InterruptedException {
59             int sum = 0;
60             for (IntWritable val : values) {
61                 sum += val.get();
62             }
63             result.set(sum);
64             context.write(key, result);
65         }
66     }
67
68     public static void main(String[] args) throws Exception {
69         Configuration conf = new Configuration();
70         String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
71         if (otherArgs.length < 2) {
72             System.err.println("Usage: wordcount <in> [<in>...]<out>");
73             System.exit(2);
74         }
75         Job job = Job.getInstance(conf, "word_count");
76         job.setJarByClass(WordCount.class);
77         job.setMapperClass(TokenizerMapper.class);
78         job.setCombinerClass(IntSumReducer.class);
79         job.setReducerClass(IntSumReducer.class);
80         job.setOutputKeyClass(Text.class);
81         job.setOutputValueClass(IntWritable.class);
82         for (int i = 0; i < otherArgs.length - 1; ++i) {
83             FileInputFormat.addInputPath(job, new Path(otherArgs[i]));
84         }
85         FileOutputFormat.setOutputPath(job,
86             new Path(otherArgs[otherArgs.length - 1]));
87         System.exit(job.waitForCompletion(true) ? 0 : 1);
88     }
89 }

```

Código 1: Código original de WordCount

```
1 package org.apache.hadoop.examples;
2
3 import java.io.IOException;
4 import java.util.StringTokenizer;
5
6 import org.apache.hadoop.conf.Configuration;
7 import org.apache.hadoop.fs.Path;
8 import org.apache.hadoop.io.IntWritable;
9 import org.apache.hadoop.io.Text;
10
11 import org.apache.hadoop.mapreduce.Job;
12 import org.apache.hadoop.mapreduce.Mapper;
13 import org.apache.hadoop.mapreduce.Reducer;
14
15 import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
16 import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
17
18 public class ExtendedWordCount {
19
20     public static class TokenizerMapper
21         extends Mapper<Object, Text, Text, IntWritable> {
22
23         private final static IntWritable one = new IntWritable(1);
24         private final Text word = new Text();
25
26         @Override
27         public void map(Object key, Text value, Context context)
28             throws IOException, InterruptedException {
29
30             String line = value.toString();
31             StringTokenizer tokenizer = new StringTokenizer(line);
32
33             while (tokenizer.hasMoreTokens()) {
34                 String token = tokenizer.nextToken();
35
36                 // 1. Convertir a minúsculas
37                 token = token.toLowerCase();
38
39                 // 2. Eliminar puntuaciones al inicio y al final (regex)
40                 token = token.replaceAll("^\\[\\p{L}\\p{Nd}]|[\\p{L}\\p{Nd}]+$", "");
41
42                 // 3. Descartar si es solo puntuación o está vacío
43                 if (token.matches("^\\[\\p{L}\\p{Nd}]+" || token.isEmpty())) {
44                     continue;
45                 }
46
47                 word.set(token);
48                 context.write(word, one);
49             }
50         }
51     }
52
53     public static class IntSumReducer
54         extends Reducer<Text, IntWritable, Text, IntWritable> {
55
56         private final IntWritable result = new IntWritable();
57
58         @Override
59         public void reduce(Text key, Iterable<IntWritable> values,
60                           Context context)
61             throws IOException, InterruptedException {
62
63             int sum = 0;
64             for (IntWritable val : values) {
65                 sum += val.get();
66             }
67
68             result.set(sum);
69             context.write(key, result);
70         }
71     }
72
73     public static void main(String[] args) throws Exception {
74         if (args.length != 2) {
75             System.err.println("Uso: ExtendedWordCount <input_path> <output_path>");
76             System.exit(-1);
77         }
78
79         Configuration conf = new Configuration();
80         Job job = Job.getInstance(conf, "extended_word_count");
81
82         job.setJarByClass(ExtendedWordCount.class);
83         job.setMapperClass(TokenizerMapper.class);
84         job.setCombinerClass(IntSumReducer.class);
85         job.setReducerClass(IntSumReducer.class);
86
87         job.setOutputKeyClass(Text.class);
88         job.setOutputValueClass(IntWritable.class);
89
90         FileInputFormat.addInputPath(job, new Path(args[0]));
91         FileOutputFormat.setOutputPath(job, new Path(args[1]));
92
93         System.exit(job.waitForCompletion(true) ? 0 : 1);
94     }
95 }
```

1.2. Implementación de selector de columna

De la misma manera que el punto anterior, me base en WordCount para crear el código 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.

```
1 package org.apache.hadoop.examples;
2
3 import java.io.IOException;
4
5 import org.apache.hadoop.conf.Configured;
6 import org.apache.hadoop.conf.Configuration;
7 import org.apache.hadoop.fs.Path;
8
9 import org.apache.hadoop.io.LongWritable;
10 import org.apache.hadoop.io.NullWritable;
11 import org.apache.hadoop.io.Text;
12
13 import org.apache.hadoop.mapreduce.Job;
14 import org.apache.hadoop.mapreduce.Mapper;
15
16 import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
17 import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
18
19 import org.apache.hadoop.util.Tool;
20 import org.apache.hadoop.util.ToolRunner;
21
22 public class SelectColumnExtractor extends Configured implements Tool {
23
24     public static class ColumnMapper extends Mapper<LongWritable, Text, NullWritable, Text> {
25         private int selectColumn = -1;
26
27         @Override
28         protected void setup(Context context) {
29             Configuration conf = context.getConfiguration();
30             selectColumn = conf.getInt("select.column", -1);
31         }
32
33         @Override
34         protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
35             String line = value.toString();
36             String[] rawColumns = line.split(",(?=\\")"); // espacio antes de comillas dobles
37             int totalColumns = rawColumns.length;
38
39             if (selectColumn >= totalColumns || selectColumn < 0) {
40                 context.write(NullWritable.get(), new Text(line));
41                 return;
42             }
43
44             String column = rawColumns[selectColumn].trim();
45             if (column.startsWith("\\")) && column.endsWith("\\"") && column.length() >= 2) {
46                 column = column.substring(1, column.length() - 1);
47             }
48
49             context.write(NullWritable.get(), new Text(column));
50         }
51     }
52
53     @Override
54     public int run(String[] args) throws Exception {
55         if (args.length != 3) {
56             System.err.println("Usage: SelectColumnExtractor <input> <output> <columnIndex>");
57             return -1;
58         }
59
60         Configuration conf = getConf();
61         conf.setInt("select.column", Integer.parseInt(args[2]));
62
63         Job job = Job.getInstance(conf, "Select_Column_Extractor");
64         job.setJarByClass(SelectColumnExtractor.class);
65
66         job.setMapperClass(ColumnMapper.class);
67         job.setNumReduceTasks(0); // solo mapper
68
69         job.setOutputKeyClass(NullWritable.class);
70         job.setOutputValueClass(Text.class);
71
72         FileInputFormat.setInputPaths(job, new Path(args[0]));
73         FileOutputFormat.setOutputPath(job, new Path(args[1]));
74
75         return job.waitForCompletion(true) ? 0 : 1;
76     }
77
78     public static void main(String[] args) throws Exception {
79         int exitCode = ToolRunner.run(new SelectColumnExtractor(), args);
80         System.exit(exitCode);
81     }
82 }
```

Código 3: Código 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 práctico ([enlace](#)).

2. Spark

A continuación 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 además 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:

```
1 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

Al igual que en las entregas anteriores del practico avanzado, el desarrollo fue desarrollado en vivo y se adjunta un video de la demostracion de este a traves de un enlace al video en YouTube ([enlace](#)).

En las revisiones de las entregas anteriores se indico que no puse evidencia suficiente, lo cual me sorprendio ya que me di el tiempo de hacer los videos explicitamente porque en la entrega incial fue recurrente el tema de la poca documentacion.

Me gusta creer que quien corrigio mis tareas no vio el link que adjunte y por ende voy a dejar en cada seccion el enlace a la parte especifica del video en donde se muestra el funcionamiento de mi codigo.

3.1. Extracción, transformación y carga

Durante el desarrollo del proceso ETL, se utilizó un clúster Spark configurado con seis máquinas tipo t3.large en AWS. Se gestionó todo el código mediante un repositorio GitHub, el cual fue clonado en el nodo master del clúster para facilitar la edición y sincronización.

El procedimiento se organizó en scripts numerados (por ejemplo, code-005.py), con scripts de ejecución asociados (por ejemplo, test-005.sh). Para comenzar, se eliminaron los datos anteriores del bucket S3 estudiantil para asegurar un entorno limpio.

El código leyó las 20 porciones del dataset desde el bucket público, aplicando el esquema correspondiente, uniendo todos los fragmentos en un único DataFrame. Se eliminaron filas con valores nulos y se tomó una muestra del 1% de los datos para pruebas. Luego, se filtraron las observaciones con category = SCIENCE y obs_type = OBJECT.

El campo template_start fue procesado para eliminar la letra T y poder convertirlo a formato Unix. Las coordenadas ecuatoriales se dividieron en grados, minutos y segundos, y se almacenó la información en formato Parquet en el bucket del estudiante bajo el nombre vlt_observations_etl.parquet.

Evidencia del desarrollo a modo de imágenes y enlace a la sección del video con la demostración ([enlace](#)).

Figura 2: Neofetch en el nodo master del cluster de Spark

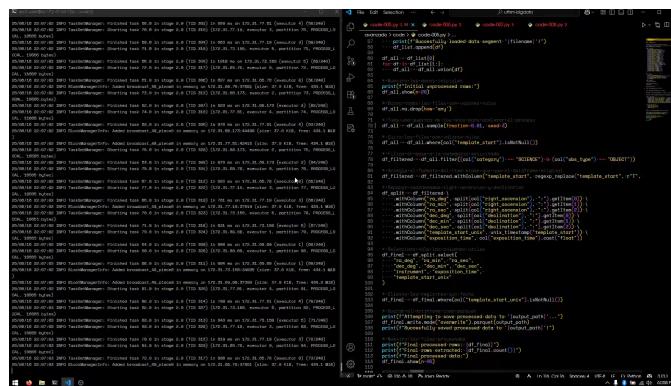


Figura 3: Spark procesando los datos de esta sección con el código al lado.

Figura 4: Dataframe inicial y resultante tras procesar los datos

```
Attempting to save processed data to 'src//284303030/int500/v1/observations.stl.parquet'...
Successfully saved processed data to 'src//284303030/int500/v1/observations.stl.parquet'
Final processed rows: dataFrame[ra_deg: string, ra_min: string, ra_sec: string, dec_deg: string, dec_min: string, dec_sec: string, instrument: string, exposition_time: float, template_start_unix: bigint]
Final rows extracted: 1363810
Final processed data:
+-----+-----+-----+-----+-----+-----+-----+-----+
| [ra_deg][ra_min][ra_sec][dec_deg][dec_min][dec_sec][instrument][exposition_time][template_start_unix] |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 05 | 57 | 07.82 | -44 | 10 | 00.4 | UVE8 | 4500.000 | 945153798 |
| 05 | 17 | 07.72 | -44 | 10 | 00.1 | UVE8 | 5400.001 | 945233260 |
| 05 | 49 | 43.41 | -38 | 10 | 33.1 | UVE8 | 4500.001 | 945315404 |
| 05 | 49 | 43.38 | -38 | 10 | 31.5 | UVE8 | 5000.001 | 945320172 |
| 05 | 17 | 07.86 | -44 | 10 | 05.6 | UVE8 | 5000.001 | 945328707 |
| 05 | 50 | 20.41 | -14 | 37 | 00.0 | UVE8 | 800.000 | 945330100 |
| 05 | 03 | 48.78 | -16 | 53 | 03.5 | UVE8 | 500.000 | 945470905 |
| 00 | 01 | 49.48 | -03 | 01 | 38.4 | UVE8 | 30.000 | 945470984 |
| 05 | 17 | 07.76 | -44 | 10 | 55.0 | UVE8 | 4500.002 | 945481494 |
| 05 | 17 | 07.76 | -44 | 10 | 55.0 | UVE8 | 4500.001 | 945481494 |
| 05 | 38 | 41.47 | -24 | 02 | 51.1 | UVE8 | 3500.999 | 945576479 |
| 05 | 17 | 07.72 | -44 | 10 | 05.9 | UVE8 | 4400.998 | 945677535 |
| 05 | 57 | 07.81 | -44 | 10 | 00.0 | UVE8 | 4500.001 | 945760331 |
| 10 | 48 | 03.05 | -59 | 41 | 00.0 | UVE8 | 00.001 | 945760453 |
| 10 | 48 | 03.05 | -59 | 41 | 00.0 | UVE8 | 1000.000 | 945760724 |
| 10 | 48 | 03.05 | -59 | 41 | 00.0 | UVE8 | 500.0 | 945760724 |
| 10 | 48 | 04.84 | -59 | 41 | 00.7 | UVE8 | 900.002 | 945763074 |
| 05 | 32 | 49.86 | -66 | 22 | 13.3 | UVE8 | 713.002 | 947562019 |
| 11 | 21 | 15.20 | -06 | 37 | 24.6 | UVE8 | 1087.001 | 947581554 |
| 05 | 32 | 49.48 | -66 | 22 | 19.9 | UVE8 | 1800.0 | 947630627 |
+-----+-----+-----+-----+-----+-----+-----+-----+
only showing top 20 rows
```

Figura 5: Dataframe resultante tras procesar los datos

```
1 from pyspark.sql import SparkSession
2 from pyspark.sql.functions import col, split, unix_timestamp, regexp_replace
3 from pyspark.sql.types import StructType, StructField, StringType
4 # Bucket personal
5 bucket = "204303630-inf356"
6 # Direccion de salida
7 output_path = f"s3a://{bucket}/vlt_observations_etl.parquet"
8 # Iniciar Spark
9 spark = SparkSession.builder.getOrCreate()
10 # Formato de los datos
11 schema = StructType([
12     StructField("object", StringType(), True),
13     StructField("right_ascension", StringType(), True),
14     StructField("declination", StringType(), True),
15     StructField("obs_timestamp", StringType(), True),
16     StructField("program_id", StringType(), True),
17     StructField("investigators", StringType(), True),
18     StructField("obs_mode", StringType(), True),
19     StructField("title", StringType(), True),
20     StructField("program_type", StringType(), True),
21     StructField("instrument", StringType(), True),
22     StructField("category", StringType(), True),
23     StructField("obs_type", StringType(), True),
24     StructField("obs_nature", StringType(), True),
25     StructField("dataset_id", StringType(), True),
26     StructField("obs_file", StringType(), True),
27     StructField("release_date", StringType(), True),
28     StructField("obs_name", StringType(), True),
29     StructField("obs_id", StringType(), True),
30     StructField("template_id", StringType(), True),
31     StructField("template_start", StringType(), True),
32     StructField("exposition_time", StringType(), True),
33     StructField("filter_lambda_min", StringType(), True),
34     StructField("filter_lambda_max", StringType(), True),
35     StructField("filter", StringType(), True),
36     StructField("grism", StringType(), True),
37     StructField("grating", StringType(), True),
38     StructField("slit", StringType(), True),
39     StructField("obs_mjd", StringType(), True),
40     StructField("airmass", StringType(), True),
41     StructField("seeing", StringType(), True),
42     StructField("distance", StringType(), True),
43     StructField("position", StringType(), True)
44 ])
```

Código 4: Código utilizado para el procedimiento de ETL

```
1 # Lee y une los segmentos del archivo original
2 df_list = []
3 for i in range(20):
4     filename = f"vlt_observations_{i:03}.csv"
5     path = f"s3a://utfsm-inf356-datasets/vlt_observations/{filename}"
6     print(f"Trying to load data segment '{filename}'...")
7     df = spark.read.csv(path, header=False, schema=schema)
8     print(f"Successfully loaded data segment '{filename}'!")
9     df_list.append(df)
10
11 df_all = df_list[0]
12 for df in df_list[1:]:
13     df_all = df_all.union(df)
14 # Muestra los datos iniciales
15 print(f"Initial unprocessed rows:")
16 df_all.show(n=20)
17 # Borra todas las filas con valores nulos
18 df_all.na.drop(how='any')
19 # Toma una muestra de los años para acelerar el proceso
20 df_all = df_all.sample(fraction=0.25, seed=3)
21 # Quita las filas con valores nulos
22 df_all = df_all.where(col("template_start").isNotNull())
23 # Filtra en base a la categoría solicitada
24 df_filtered = df_all.filter((col("category") == "SCIENCE") & (col("obs_type"
25     ) == "OBJECT"))
26 # Arregla el formato del time stamp que usa el dataframe original
27 df_filtered = df_filtered.withColumn("template_start", regexp_replace(
28     "template_start", r"T", " "))
29 # Separar coordenadas right ascension y declination
30 df_split = df_filtered \
31     .withColumn("ra_deg", split(col("right_ascension"), ":").getItem(0)) \
32     .withColumn("ra_min", split(col("right_ascension"), ":").getItem(1)) \
33     .withColumn("ra_sec", split(col("right_ascension"), ":").getItem(2)) \
34     .withColumn("dec_deg", split(col("declination"), ":").getItem(0)) \
35     .withColumn("dec_min", split(col("declination"), ":").getItem(1)) \
36     .withColumn("dec_sec", split(col("declination"), ":").getItem(2)) \
37     .withColumn("template_start_unix", unix_timestamp("template_start")) \
38     .withColumn("exposition_time", col("exposition_time").cast("float"))
```

Código 5: Código utilizado para el procedimiento de ETL

```
1 # Selecciona solo las columnas utiles
2 df_final = df_split.select(
3     "ra_deg", "ra_min", "ra_sec",
4     "dec_deg", "dec_min", "dec_sec",
5     "instrument", "exposition_time",
6     "template_start_unix"
7 )
8 # Elimina los registros sin fecha
9 df_final = df_final.where(col("template_start_unix").isNotNull())
10 # Guarda el dataframe como parquet
11 print(f"Attempting to save processed data to '{output_path}'...")
12 df_final.write.mode("overwrite").parquet(output_path)
13 print(f"Successfully saved processed data to '{output_path}'!")
14 # Muestra las filas procesadas
15 print(f"Final processed rows: {df_final}")
16 print(f"Final rows extracted: {df_final.count()}")
17 print(f"Final processed data:")
18 df_final.show(n=20)
19 # Detiene Spark
20 spark.stop()
```

Código 6: Código utilizado para el procedimiento de ETL

3.2. Coordenadas galácticas

Se desarrolló un programa que transforma las coordenadas ecuatoriales en galácticas a partir del archivo generado en el proceso ETL. Se cargó el archivo `vlt_observations_etl.parquet`, tomando una muestra del 25 % para agilizar el desarrollo.

Se convirtieron las coordenadas de grados, minutos y segundos a grados decimales utilizando cálculos trigonométricos básicos. Luego, se realizaron las transformaciones necesarias para obtener la ascensión recta y declinación galáctica, las cuales fueron almacenadas junto con los campos `instrument`, `exposition_time` y `template_start`.

El resultado se guardó en el archivo `vlt_observations_gc.parquet`.

Evidencia del desarrollo a modo de imágenes y enlace a la sección del video con la demostración ([enlace](#)).

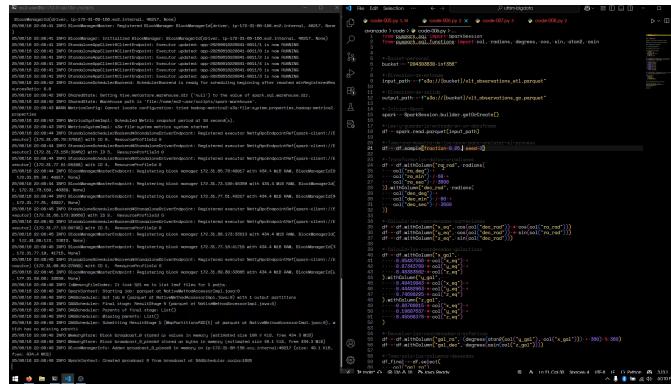


Figura 6: Spark procesando los datos mientras explico el código

```
Using temporary ARW credentials
-- loading settings: url = jar://file:/home/ec2-user/spark/jars/ivy-2.5.1.jar!/org/apache/ivy/core/settings/ivysettings.xml
Final processed rows: DataFrame[gcl_rt: double, gcl_dec: double, instrument: string, exposition_time: float, template_start_unix: bigint
Final processed rows extracted: 325122
Final processed data:
+-----+-----+-----+-----+
|    gcl_rt|   gcl_dec|instrument|exposition_time|template_start_unix|
+-----+-----+-----+-----+
  122.2321249657035986 | -53.263495719419 | VIRCAM | 60.0 | 13382952941 |
  145.67419820530891 | -67.2409423839679 | OMEGACAM | 45.0 | 133829529282 |
  145.92192100000001 | -64.82000000000001 | VIRCAM | 60.0 | 133829529161 |
  145.92192100000001 | -70.2147015313 | VIRCAM | 15.0 | 133829529355 |
  203.88684313920991 | -73.323215383033 | CHIRES | 30.0 | 133828349466 |
  208.9862900081158203 | -73.237508553601 | CHIRES | 30.0 | 133828349466 |
  334.339488018158704 | -67.67199158657 | OMEGACAM | 45.0 | 133828456789 |
  268.7675889708567 | -82.61991576685404 | NOAO+CONICA | 30.0 | 133820640407 |
  268.774380518158245 | -82.6151688438082 | NOAO+CONICA | 30.0 | 133820640407 |
  106.0124040532351 | -47.2066822009082 | OMEGACAM | 45.0 | 133820640477 |
  148.54968188268957 | -67.23844577908264 | OMEGACAM | 45.0 | 133828890902 |
  100.776228860918207 | -90.09422441082862 | ISAAC | 76.0 | 133826429296 |
  199.19457291421889 | -95.29713077032891 | VIRCAM | 4.0 | 133826743930 |
  199.19457291421889 | -95.29713077032891 | NOAO+CONICA | 21.0 | 133826743930 |
  199.19457291421889 | -95.29713077032891 | VIRCAM | 21.0 | 133826743930 |
  199.19457291421889 | -95.29713077032891 | NOAO+CONICA | 21.0 | 133826743930 |
  219.13830411180881 | -37.37919584117307 | VIRCAM | 4.0 | 133826771718 |
  199.19457291421889 | -95.29713077032891 | NOAO+CONICA | 21.0 | 133826771718 |
  221.99836953759891 | -35.958675582982 | VIRCAM | 4.0 | 133826771718 |
  221.18687938376943 | -85.351973485690779 | VIRCAM | 4.0 | 133826771718 |
  179.11345428713584 | -84.1729807340016 | VIRCAM | 4.0 | 133826806161 |
+-----+-----+-----+-----+
only showing 20 rows
```

Figura 7: Dataframe resultante tras procesar los datos

```
1 from pyspark.sql import SparkSession
2 from pyspark.sql.functions import col, radians, degrees, cos, sin, atan2,
3     ↪ asin
4 # Bucket personal
5 bucket = "204303630-inf356"
6 # Direccion de entrada
7 input_path = f"s3a://{bucket}/vlt_observations_etl.parquet"
8 # Direccion de salida
9 output_path = f"s3a://{bucket}/vlt_observations_gc.parquet"
10 # Iniciar Spark
11 spark = SparkSession.builder.getOrCreate()
12 # Lee y guarda la entrada en un dataframe
13 df = spark.read.parquet(input_path)
14 # Toma una muestra de los anos para acelerar el proceso
15 df = df.sample(fraction=0.25, seed=3)
16 # Transforma los datos a radianes
17 df = df.withColumn("ra_rad", radians(
18     col("ra_deg") +
19     col("ra_min") / 60 +
20     col("ra_sec") / 3600
21 ).withColumn("dec_rad", radians(
22     col("dec_deg") +
23     col("dec_min") / 60 +
24     col("dec_sec") / 3600
25 ))
26 # Calcula las coordenadas cartesianas
27 df = df.withColumn("x_eq", cos(col("dec_rad")) * cos(col("ra_rad")))
28 df = df.withColumn("y_eq", cos(col("dec_rad")) * sin(col("ra_rad")))
29 df = df.withColumn("z_eq", sin(col("dec_rad")))
```

Código 7: Código utilizado para el cálculo de coordenadas galácticas

```
1 # Calcula las coordenadas galácticas
2 df = df.withColumn("x_gal",
3     -0.05487556 * col("x_eq") +
4     -0.87343709 * col("y_eq") +
5     -0.48383502 * col("z_eq")
6 ).withColumn("y_gal",
7     0.49410943 * col("x_eq") +
8     -0.44482963 * col("y_eq") +
9     0.74698225 * col("z_eq")
10 ).withColumn("z_gal",
11     -0.86766615 * col("x_eq") +
12     -0.19807637 * col("y_eq") +
13     0.45598378 * col("z_eq")
14 )
15 # Devuelve las coordenadas a esféricas
16 df = df.withColumn("gal_ra", (degrees(atan2(col("y_gal"), col("x_gal")))) +
17     ↪ 360) % 360)
17 df = df.withColumn("gal_dec", degrees(asin(col("z_gal"))))
18 # Toma sola las columnas deseadas
19 df_final = df.select(
20     col("gal_ra"),
21     col("gal_dec"),
22     col("instrument"),
23     col("exposition_time"),
24     col("template_start_unix")
25 )
26 # Guarda el dataframe como parquet
27 df_final.write.mode("overwrite").parquet(output_path)
28 # Muestra las filas procesadas
29 print(f"Final\u00d7processed\u00d7rows:\u00d7{df_final}")
30 print(f"Final\u00d7rows\u00d7extracted:\u00d7{df_final.count()}")
31 print(f"Final\u00d7processed\u00d7data:")
32 df_final.show(n=20)
33 # Detiene Spark
34 spark.stop()
```

Código 8: Código utilizado para el cálculo de coordenadas galácticas

3.3. Segmentación temporal

A partir del archivo con coordenadas galácticas, se segmentaron los datos por año y por semana del año, considerando como primera semana la que contiene el primer lunes. Las fechas anteriores a ese lunes fueron asociadas al año anterior.

Se transformó la fecha desde formato Unix a calendario, se identificaron los años distintos y el primer lunes correspondiente para cada uno. Luego, se asignaron las observaciones a su semana correspondiente y se ajustó el año cuando era necesario.

Finalmente, los datos fueron almacenados en formato Parquet dentro del directorio `partition/`, organizados por carpeta de año. Cada año contiene un archivo con todos sus datos (`vlt_observations_XXXX.parquet`) y una subcarpeta `weeks/` con archivos semanales por año.

Es importante recalcar que justamente al ejecutar este código, AWS se colapsó y me echo del cluster de Spark, e incluso en el video se muestra como no se alcanzó a procesar toda la data.

Ya que este procedimiento era bastante largo, decidí revisar el bucket para ver si proceso algo de los datos entregados, los nodos alcanzaron a agrupar solo las semanas del 2018, sin embargo se podía ver que se cumplía el formato pedido de las fechas y numeración de semanas, por lo que decidí dejarlo hasta ahí y seguir con el punto siguiente.

- Ejecución y mi explicación del código: [enlace](#).
- Momento en donde AWS me echa del entorno: [enlace](#).
- Evidencia de que los datos se guardaron: [enlace](#).

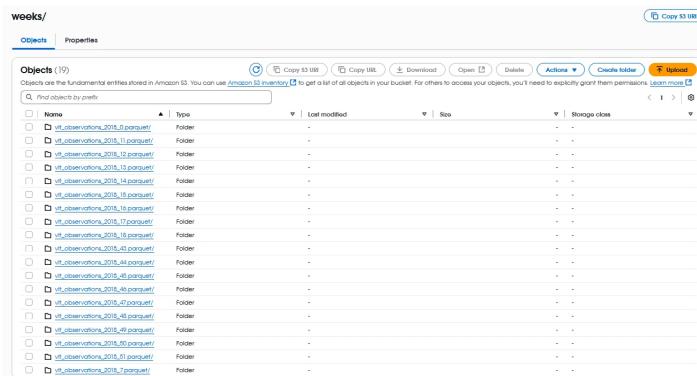


Figura 8: Pantallazo del Bucket S3 resultante tras procesar los datos

```
1 from pyspark.sql import SparkSession
2 from pyspark.sql.functions import col, from_unixtime, to_date, date_format,
3     → unix_timestamp, expr, floor, min
4 from pyspark.sql.window import Window
5 # Bucket personal
6 bucket = "204303630-inf356"
7 # Direccion de entrada
8 input_path = f"s3a://{bucket}/vlt_observations_gc.parquet"
9 # Direccion de la salida
10 output_root = f"s3a://{bucket}/partition"
11 # Iniciar Spark
12 spark = SparkSession.builder.getOrCreate()
13 # Lee el archivo con coordenadas galácticas
14 df = spark.read.parquet(input_path)
15 # Convierte la fecha unix a fecha date
16 df = df.withColumn("obs_date", to_date(from_unixtime(col("→ template_start_unix")))))
17 # Calcula el primer lunes anual
18 years_df = df.selectExpr("year(obs_date) as year").distinct()
19 print(f"All distinct years read from data: {years_df}")
20 years_df = years_df.withColumn(
21     "first_lmonday",
22     expr("""→
23         next_day(to_date(concat(year, '-01-01')), 'Monday')
24         """))
25 print(f"Adding column to data for the first monday of the year: {years_df}")
26 # Asocia cada fila al primer lunes anual
27 df = df.withColumn("year", expr("year(obs_date)"))
28 print(f"Adding new column for the year: {df}")
29 df = df.join(years_df, on="year", how="left")
30 print(f"Associate each row to first year monday: {df}")
```

Código 9: Código utilizado para segmentación temporal de datos

```

1 # Calcula la semana desde el primer lunes
2 df = df.withColumn(
3     "week",
4     floor(
5         (unix_timestamp(col("obs_date")) - unix_timestamp(col("first_lmonday"
6             ↪ "))) /
7             (86400 * 7)
8     )
9 )
10 print(f"Calculating weeks since first monday of the year: {df}")
11 # Ajusta al anterior anterior si la fecha es antes del primer lunes
12 df = df.withColumn(
13     "adjusted_year",
14     expr("IF(obs_date < first_lmonday, year - 1, year)"))
15 ).withColumn(
16     "adjusted_week",
17     expr("IF(obs_date < first_lmonday, 0, week)"))
18 )
19 print(f"Adding days before the first monday to the year before: {df}")
20 # Toma una muestra de los años para acelerar el proceso
21 df = df.sample(fraction=0.25, seed=3)
22 # Escribir archivo anuales
23 for year in df.select("adjusted_year").distinct().collect():
24     y = year["adjusted_year"]
25     df_year = df.filter(col("adjusted_year") == y).drop("first_lmonday", "year", "week")
26     df_year.write.mode("overwrite").parquet(f"{output_root}/{y}/"
27             ↪ vlt_observations_{y}.parquet")
28     # Subdivide por semana para agrupar anualmente
29     df_weeks = df_year.withColumn("week", col("adjusted_week"))
30     # Toma una muestra de las semanas para acelerar el proceso
31     df_sample = df_weeks.sample(fraction=0.25, seed=3)
32     for week in df_sample.select("week").distinct().collect():
33         w = week["week"]
34         df_w = df_sample.filter(col("week") == w)
35         df_w.write.mode("overwrite").parquet(
36             f"{output_root}/{y}/weeks/vlt_observations_{y}_{w}.parquet"
37         )
38         print(f"Final processed data for week {w} of the year {y}:")
39         df_w.show(n=20)
# Detiene Spark
spark.stop()

```

Código 10: Código utilizado para segmentación temporal de datos

3.4. Conteo de observaciones

El último paso fue generar un conteo de observaciones agrupadas por instrumento y por bloques de 10 grados del cielo tanto horizontal como verticalmente. A partir del archivo con coordenadas galácticas, se agruparon los datos según el centro de cada celda angular de 10x10 grados.

En cada celda, se sumó el tiempo total de exposición en lugar de mantener los valores individuales, y se descartó el campo `template_start`. El archivo de salida se guardó con el mismo nombre que el archivo de entrada, pero agregando `.count` antes de la extensión.

Evidencia del desarrollo a modo de images y enlace a la sección del video con la demostración ([enlace](#)).

```
Using temporary AWS credentials
-- loading settings :: url = jar:file:/home/ec2-user/spark/jars/ivy-2.5.1.jar!/org/apache/ivy/core/settings/ivysettings.xml
[INFO] Registring ivy-2.5.1.jar
[INFO] Attaching artifact ivy-2.5.1.jar
[INFO] Saving processed data to "s3a://204303630-inf356/vit_observations.go.count.parquet"
[INFO] Successfully saved processed data to "s3a://204303630-inf356/vit_observations.go.count.parquet"
[INFO] Final processed rows: Dataframe[gol.ra: bigint, gol.dec: bigint, instrument: string, observation_count: bigint, total_exposition_time: double]
[INFO] Final rows extracted: 1871
[INFO] Final processed data:
+-----+-----+-----+-----+
|gol.ra|gol.dec|instrument|observation_count|total_exposition_time|
+-----+-----+-----+-----+
| 135| -45|NAOS-CONICA|      679| 23208.271052508983|
| 140| -65| XSHOOTER|      120| 66588.448951171891|
| 145| -65| HAWKIE|      125| 70000.000000000000|
| 140| -65| MATTSE|       85| 53999.000000000000|
| 125| -65| VIMOS|      635| 329994.5459668292|
| 135| -45| MUSE|       36| 22045.0|
| 65| -65| SPHERE|       23| 722.83700001039781|
| 270| -65| HAWKIE|       80| 25000.000000000000|
| 100| -65| CAMELEON|      120| 25000.144999999997|
| 115| -65| NAOS-CONICA|      334| 21024.107998881538|
| 105| -45| NAOS-CONICA|       90| 4916.4890000424989|
| 315| -65| VIMOS|       23| 20360.02082824797|
| 200| -65| OMEGACAM|      103| 10999.0|
| 250| -75| NAOS-CONICA|      100| 21334.847999999997|
| 65| -75| NAOS-CONICA|      66| 2656.0089803738463|
| 55| -65| MUSE|       34| 31679.26397705878|
| 125| -65| HAWKIE|      344| 31285.09700012297|
| 170| -65| FOR82|       21| 15337.7431568894|
| 345| -65| VIRCAM|      531| 10740.5|
| 130| -75| SPHERE|      62| 2718.3109888898900|
+-----+
only showing top 20 rows
```

Figura 9: Dataframe resultante tras procesar los datos

```

1 from pyspark.sql import SparkSession
2 from pyspark.sql.functions import col, floor, avg, count, sum as sum_, lit
3 # Bucket personal
4 bucket = "204303630-inf356"
5 # Direccion de entrada
6 input_file = "vlt_observations_gc.parquet"
7 input_path = f"s3a://{bucket}/{input_file}"
8 # Direccion de la salida
9 output_file = input_file.replace(".parquet", ".count.parquet")
10 output_path = f"s3a://{bucket}/{output_file}"
11 # Iniciar Spark
12 spark = SparkSession.builder.getOrCreate()
13 # Leer archivo
14 df = spark.read.parquet(input_path)
15 print(f"Total registers loaded: {df.count()}")
16 # Asegurarse que las coordenadas están en float
17 df = df.withColumn("gal_ra", col("gal_ra").cast("float"))
18 df = df.withColumn("gal_dec", col("gal_dec").cast("float"))
19 df = df.withColumn("exposition_time", col("exposition_time").cast("float"))
20 # Crear segmentos de 10 grados
21 df = df.withColumn("ra_bin", floor(col("gal_ra") / 10) * 10 + 5) # Centro
    ↪ del bin
22 df = df.withColumn("dec_bin", floor(col("gal_dec") / 10) * 10 + 5)
23 # Agrupar por bin e instrumento
24 df_grouped = df.groupBy("ra_bin", "dec_bin", "instrument").agg(
25     count("*").alias("observation_count"),
26     sum_("exposition_time").alias("total_exposition_time")
27 )
28 # Renombrar columnas para mayor claridad
29 df_grouped = df_grouped.withColumnRenamed("ra_bin", "gal_ra") \
    .withColumnRenamed("dec_bin", "gal_dec")
30 # Guardar resultado
31 print(f"Attempting to save processed data to '{output_path}'...")
32 df_grouped.write.mode("overwrite").parquet(output_path)
33 print(f"Successfully saved processed data to '{output_path}'!")
34 # Muestra las filas procesadas
35 print(f"Final unprocessed rows: {df_grouped.count()}")
36 print(f"Final rows extracted: {df_grouped.count()}")
37 print(f"Final unprocessed data:")
38 df_grouped.show(n=20)
39 # Detener Spark
40 spark.stop()

```

Código 11: Código utilizado para segmentación temporal de datos