



UNIVERSIDAD TECNICA  
FEDERICO SANTA MARIA

# Trabajo Práctico 2

INF356 - 2025-1 - 200

Computación Distribuida para Big Data

1 de junio de 2025 - v1.0

## Índice

<b>1. Map Reduce</b>	<b>4</b>
1.1. Implementación extendida de wordcounter . . . . .	5
1.2. Implementación de selector de columna . . . . .	8
1.3. Tiempo de ejecución . . . . .	10
<b>2. Spark</b>	<b>11</b>
2.1. Uso de training-bigdata-002 . . . . .	11
2.2. Escalamiento vertical y horizontal . . . . .	11
<b>3. Procesamiento de datos</b>	<b>13</b>
3.1. Extracción, transformación y carga . . . . .	13
3.2. Coordenadas galácticas . . . . .	14
3.3. Segmentación temporal . . . . .	15
3.4. Conteo de observaciones . . . . .	16

## Índice de tablas

1. Tiempos de ejecución del WordCount extendido . . . . .	10
2. Tiempos de ejecución del SelectColumnExtractor . . . . .	10
3. Tiempos de ejecución para medición de impacto de escalamiento . . . . .	12

## Índice de figuras

1. Documentacion oficial de Oracle sobre como parchar un JAR . . . . .	5
--	---

2.	Efecto del escalamiento vertical y horizontal en el tiempo de ejecución de test-001.sh <b>Tiene libertad para elegir la metodología de visualización que le parezca más apropiada</b> . . . . .	13
----	---	----

**Índice de fragmentos de codigo**

1.	Codigo original de WordCount . . . . .	6
2.	Codigo de ExtendedWordCount . . . . .	7
3.	Codigo de SelectColumnExtractor . . . . .	9
4.	Código utilizado para el procedimiento de ETL . . . . .	14
5.	Código utilizado para el cálculo de coordenadas galácticas . . . . .	15
6.	Código utilizado para segmentación temporal de datos . . . . .	16
7.	Código utilizado para segmentación temporal de datos . . . . .	16

## 1. Map Reduce

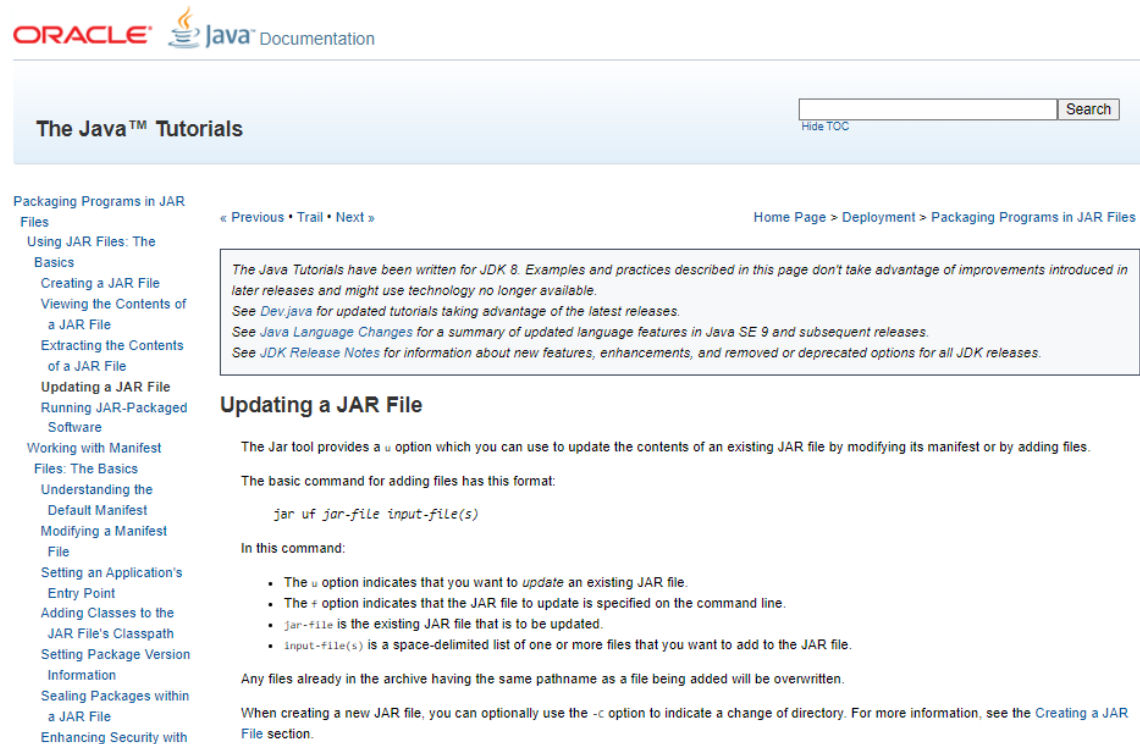
Esta entrega consistia en modificar el codigo fuente de Hadoop para agregar funcionalidad extra, en donde habia que implementar metodos a partir de la funcion WordCount.


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.



**ORACLE**  **java** Documentation

## The Java™ Tutorials

« Previous • Trail • Next » Home Page > Deployment > Packaging Programs in JAR Files

**Packaging Programs in JAR Files**

- Files
  - Using JAR Files: The Basics
    - Creating a JAR File
    - Viewing the Contents of a JAR File
    - Extracting the Contents of a JAR File
    - Updating a JAR File
    - Running JAR-Packaged Software
  - Working with Manifest
  - Files: The Basics
    - Understanding the Default Manifest
    - Modifying a Manifest File
    - Setting an Application's Entry Point
    - Adding Classes to the JAR File's Classpath
    - Setting Package Version Information
    - Sealing Packages within a JAR File
    - Enhancing Security with

*The Java Tutorials have been written for JDK 8. Examples and practices described in this page don't take advantage of improvements introduced in later releases and might use technology no longer available. See [Dev.java](#) for updated tutorials taking advantage of the latest releases. See [Java Language Changes](#) for a summary of updated language features in Java SE 9 and subsequent releases. See [JDK Release Notes](#) for information about new features, enhancements, and removed or deprecated options for all JDK releases.*

### Updating a JAR File

The Jar tool provides a `u` option which you can use to update the contents of an existing JAR file by modifying its manifest or by adding files.

The basic command for adding files has this format:

```
jar uf jar-file input-file(s)
```

In this command:

- The `u` option indicates that you want to *update* an existing JAR file.
- The `+` option indicates that the JAR file to update is specified on the command line.
- `jar-file` is the existing JAR file that is to be updated.
- `input-file(s)` is a space-delimited list of one or more files that you want to add to the JAR file.

Any files already in the archive having the same pathname as a file being added will be overwritten.

When creating a new JAR file, you can optionally use the `-c` option to indicate a change of directory. For more information, see the [Creating a JAR File](#) section.

**Figura 1:** Documentación oficial de Oracle sobre como parchar un JAR

## 1.1. Implementación extendida de wordcounter

Para desarrollar esta nueva funcionalidad me base en la implementación de la clase original de WordCount, la cual obtuve desde el repositorio público de Apache Hadoop en GitHub ([enlace](#)), en donde dicha función se encuentra en la carpeta de ejemplos.

La forma en la que luego parche el código, fue aprovechándome de los .jar que contiene el binario de Hadoop, estos últimos 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 continuación dejare el código original de WordCount y el código 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 índice/numero de columna. Esta función 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("(?=\\s)"); // 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 practico ([enlace](#)).

## 2. Spark

### 2.1. Uso de training-bigdata-002

Demuestre que ha logrado crear un cluster de spark, testear el cluster y destruir el cluster de acuerdo a las instrucciones entregadas en <https://github.com/ptoledo-teaching/training-bigdata-002>

### 2.2. Escalamiento vertical y horizontal

Modificando el archivo de configuración de deployment del cluster de Spark, debe desplegar 5 configuraciones para medir el impacto que tienen el escalamiento vertical y horizontal en el tiempo de procesamiento. A cada estudiante le corresponderá una serie de configuraciones diferentes que se puede obtener de la siguiente lista:

- **19500967-8:** 2 x t3.large, 4 x t3.small, 8 x t3.large, 8 x t3.large, 8 x t3.small, 4 x t3.medium, 6 x t3.micro, 8 x t3.large, 2 x t3.micro, 4 x t3.medium
- **20068377-3:** 6 x t3.micro, 6 x t3.micro, 2 x t3.micro, 6 x t3.large, 10 x t3.small, 2 x t3.micro, 2 x t3.large, 2 x t3.medium, 10 x t3.large, 2 x t3.micro
- **20241011-1:** 4 x t3.micro, 10 x t3.micro, 8 x t3.micro, 4 x t3.large, 8 x t3.medium, 10 x t3.medium, 6 x t3.medium, 4 x t3.large, 4 x t3.medium, 6 x t3.medium
- **20298815-6:** 2 x t3.small, 10 x t3.medium, 10 x t3.large, 8 x t3.medium, 6 x t3.small, 6 x t3.medium, 2 x t3.micro, 2 x t3.small, 6 x t3.small, 2 x t3.large
- **20430363-0:** 4 x t3.micro, 4 x t3.small, 10 x t3.medium, 6 x t3.large, 8 x t3.medium, 10 x t3.micro, 4 x t3.small, 8 x t3.small, 10 x t3.large, 8 x t3.medium
- **20632334-5:** 6 x t3.medium, 4 x t3.medium, 4 x t3.medium, 6 x t3.large, 8 x t3.micro, 6 x t3.small, 10 x t3.large, 10 x t3.large, 4 x t3.small, 2 x t3.micro
- **20762701-1:** 8 x t3.micro, 2 x t3.medium, 6 x t3.micro, 10 x t3.small, 10 x t3.small, 10 x t3.medium, 4 x t3.large, 4 x t3.medium, 2 x t3.small, 2 x t3.large
- **20781979-4:** 10 x t3.large, 6 x t3.medium, 6 x t3.large, 2 x t3.small, 10 x t3.small, 2 x t3.medium, 10 x t3.small, 4 x t3.small, 8 x t3.medium, 8 x t3.large
- **20886083-6:** 2 x t3.large, 6 x t3.large, 4 x t3.micro, 8 x t3.micro, 4 x t3.large, 6 x t3.micro, 4 x t3.micro, 6 x t3.large, 6 x t3.large, 4 x t3.micro
- **20920259-K:** 4 x t3.small, 8 x t3.micro, 2 x t3.large, 8 x t3.medium, 6 x t3.micro, 8 x t3.small, 8 x t3.large, 2 x t3.small, 10 x t3.large, 4 x t3.micro

Maquinas	Tipo	Tiempo[s]	Costo[USD]

**Tabla 3:** Tiempos de ejecución para medición de impacto de escalamiento

- **20966993-5:** 8 x t3.small, 6 x t3.small, 2 x t3.small, 10 x t3.large, 4 x t3.medium, 4 x t3.micro, 6 x t3.medium, 8 x t3.medium, 6 x t3.small, 8 x t3.micro
- **20969314-3:** 4 x t3.large, 10 x t3.small, 8 x t3.micro, 4 x t3.large, 4 x t3.medium, 10 x t3.micro, 10 x t3.micro, 8 x t3.small, 8 x t3.large, 4 x t3.large
- **21095788-K:** 4 x t3.micro, 4 x t3.small, 10 x t3.medium, 2 x t3.medium, 2 x t3.micro, 6 x t3.medium, 6 x t3.large, 10 x t3.small, 8 x t3.medium, 6 x t3.large
- **21127094-2:** 4 x t3.large, 6 x t3.small, 2 x t3.large, 2 x t3.small, 8 x t3.small, 2 x t3.medium, 10 x t3.medium, 2 x t3.small, 8 x t3.small, 10 x t3.medium
- **26799666-0:** 2 x t3.medium, 10 x t3.micro, 6 x t3.small, 2 x t3.medium, 10 x t3.micro, 2 x t3.medium, 6 x t3.small, 2 x t3.micro, 2 x t3.medium, 8 x t3.large

La cantidad de máquinas se refiere a la cantidad de máquinas trabajadoras. Usted debe desplegar el cluster para cada una de las configuraciones que le corresponden, ejecutar el test-000 (esto es necesario ya que este paso instala ciertas librerías que tienen un impacto relevante en el tiempo de ejecución) y luego medir el tiempo que demora el test-001. El tiempo debe ser reportado en <https://forms.gle/gGVc2wkqc6FzfyU99>. Debe completar el formulario para cada una de las 5 configuraciones solicitadas. En los casos de quienes tienen 2 veces asignada la misma configuración, se debe medir 2 veces y reportar 2 veces el tiempo requerido para ejecución.

Con la información obtenida se debe completar la tabla 3. En la columna de **costo** debe calcular el costo que tuvo la ejecución en USD tomando como referencia los precios on-demand por hora disponibles en <https://aws.amazon.com/ec2/instance-types/t3/>.

Los datos recopilados entre todos los estudiantes estarán disponibles en [https://docs.google.com/spreadsheets/d/1t53S2s0knpQnZl9EcZr2ZHW0g\\_qTCcNp6Kjpg7g1Wlo/edit?usp=sharing](https://docs.google.com/spreadsheets/d/1t53S2s0knpQnZl9EcZr2ZHW0g_qTCcNp6Kjpg7g1Wlo/edit?usp=sharing). En base a esta información usted debe desarrollar una figura que permita apreciar el efecto multidimensional del escalamiento vertical y horizontal en el tiempo de ejecución del proceso. Debe considerar a lo menos 10 pares diferentes maquina/cantidad para poder desarrollar esta figura (deberá esperar a que exista esta cantidad mínima de información para poder proceder).



**Figura 2:** Efecto del escalamiento vertical y horizontal en el tiempo de ejecución de test-001.sh **Tiene libertad** para elegir la metodología de visualización que le parezca más apropiada

Discuta sobre como afecta el escalamiento horizontal y vertical el tiempo de proceso del test-001, considerando tanto a la información que recopiló con sus experimentos, la información obtenida desde los experimentos de los otros participantes de la asignatura, y el impacto de la dispersión de las mediciones.

### 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](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.

```
1 from pyspark.sql import SparkSession
2
3 # Create a SparkSession
4 spark = SparkSession.builder.getOrCreate()
5
6 # Stop Spark session
7 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.

```
1 from pyspark.sql import SparkSession
2
3 # Create a SparkSession
4 spark = SparkSession.builder.getOrCreate()
5
6 # Stop Spark session
7 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 raíz 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.

```
1 from pyspark.sql import SparkSession
2
3 # Create a SparkSession
4 spark = SparkSession.builder.getOrCreate()
5
6 # Stop Spark session
7 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.

```
1 from pyspark.sql import SparkSession
2
3 # Create a SparkSession
4 spark = SparkSession.builder.getOrCreate()
5
6 # Stop Spark session
7 spark.stop()
```

**Código 7:** Código utilizado para segmentación temporal de datos