INFB8090 - COMPUTACION PARALELA Y DISTRIBUIDA

INGENIERÍA CIVIL EN CIENCIA DE DATOS

Informe° 1

“ Un problema de transporte ”

SEBASTIAN SALAZAR MOLINA

Julio de 2025

UNIVERSIDAD TECNOLÓGICA METROPOLITANA
UTEM del Estado de Chile

INFORME° 1

INTEGRANTES:

Nombre: Glenn Lanyon Alarcon **Email:** glanyon@utem.cl
Nombre: Felipe Villa Miranda **Email:** fvilla@utem.c
Nombre: Javier Fernandez **Email:** jfernandezy@utem.cl

1. Resumen Ejecutivo

El presente trabajo aborda un problema de análisis masivo de datos demográficos y de movilidad en el mundo ficticio de Eldoria, con un énfasis central en la aplicación de computación paralela para el procesamiento eficiente de grandes volúmenes de información. El objetivo general fue identificar los 10.000 trayectos más frecuentes entre poblados (definidos por los códigos postales de origen y destino), con el propósito de orientar decisiones estratégicas en infraestructura vial y transporte.

La metodología implementada se basó en el uso de PySpark, una herramienta distribuida y paralela que permite la ejecución de operaciones sobre datasets de gran escala mediante la partición de datos y el procesamiento concurrente. El código desarrollado ejecuta agrupamientos (groupBy), conteos (count) y ordenamientos (orderBy) de manera paralela sobre un dataset de 100 millones de registros, lo que sería impracticable con bibliotecas tradicionales como pandas debido a limitaciones de memoria y rendimiento.

En cuanto a los resultados principales, se logró generar un ranking con los 10.000 trayectos más frecuentes del reino, información crítica para la planificación territorial. Aunque se presentaron inconvenientes relacionados con la exportación en entornos Windows por configuración de HADOOPHOME, se aplicó una solución intermedia utilizando pandas solo para el almacenamiento, manteniendo el procesamiento de datos principal en PySpark.

La conclusión clave es que el uso de herramientas de computación paralela como PySpark permite abordar problemas de alta demanda computacional de forma escalable, eficiente y reproducible. El proyecto demuestra cómo el procesamiento distribuido puede aplicarse en contextos prácticos, reduciendo significativamente los tiempos de cómputo y aprovechando al máximo la arquitectura multicore disponible en el entorno de trabajo.

2. Introducción

En el contexto actual de análisis de datos a gran escala, el manejo eficiente de información masiva se ha convertido en un desafío crítico para la toma de decisiones. Este informe se sitúa dentro del ámbito de la computación paralela y distribuida, una disciplina que permite procesar grandes volúmenes de datos mediante técnicas que aprovechan múltiples núcleos de procesamiento y entornos de ejecución escalables. En particular, se aborda el análisis de un set de datos censales y de movilidad del mundo ficticio de Eldoria, compuesto por aproximadamente 100 millones de registros, cuya magnitud exige el uso de estrategias computacionales avanzadas.

El objetivo principal del informe es aplicar principios de computación paralela para identificar los 10.000 trayectos más frecuentes entre zonas geográficas del territorio de Eldoria, entendiendo cada trayecto como una combinación de código postal de origen y destino. Este análisis no solo demuestra la capacidad técnica del procesamiento distribuido, sino que también provee información estratégica útil para la planificación territorial y la optimización del sistema de transporte en escenarios simulados.

La justificación de este trabajo radica en la necesidad de utilizar herramientas modernas como PySpark para resolver problemas de escala que serían inabordables con técnicas secuenciales tradicionales.

La capacidad de distribuir tareas, minimizar tiempos de respuesta y manejar datos en paralelo lo convierte en un enfoque esencial tanto en entornos académicos como industriales. Además, el uso de un caso ficticio permite experimentar libremente con técnicas de paralelización sin comprometer datos sensibles.

En cuanto al alcance, el informe se concentra exclusivamente en el procesamiento de los datos de movilidad entre poblados, dejando fuera otros análisis posibles como la visualización geoespacial, el estudio por especie o género, o la inferencia sobre patrones demográficos. Tampoco se considera el entrenamiento de modelos predictivos, ya que el foco está puesto en la manipulación y análisis exploratorio paralelo de los datos a gran escala.

3. Marco Teórico

La computación paralela es una rama de la informática que se centra en la ejecución simultánea de múltiples procesos con el objetivo de reducir los tiempos de cómputo y aumentar la eficiencia en el manejo de grandes volúmenes de datos. Este paradigma es particularmente relevante cuando se trabaja con Big Data, donde los métodos tradicionales de procesamiento secuencial resultan ineficaces o incluso inviables debido a limitaciones de memoria y tiempo de ejecución.

3.1 Modelo MapReduce y Apache Spark

Uno de los modelos más ampliamente utilizados en la computación paralela distribuida es *MapReduce*, propuesto inicialmente por Google, que permite dividir un problema en pequeñas tareas independientes ("Map") que luego se combinan ("Reduce"). En este proyecto, dicha lógica fue implementada de forma práctica mediante el uso de **Apache Spark**, una plataforma de análisis distribuido en memoria, que se ejecuta sobre el modelo de ejecución paralela y que permite trabajar eficientemente con grandes conjuntos de datos.

3.2 PySpark y su Rol en la Computación Paralela

PySpark, la interfaz de Python para Apache Spark, permite manipular estructuras de datos distribuidas llamadas *DataFrames*, que se procesan en paralelo en múltiples nodos o hilos de ejecución. En el código desarrollado, PySpark fue empleado para:

- Leer un archivo CSV masivo utilizando un plan de ejecución paralelo (`inferSchema=True` y `header=True` para un reconocimiento eficiente de estructura).
- Agrupar datos por combinación de códigos postales (`groupBy("CP ORIGEN", "CP DESTINO")`) y aplicar una operación de conteo distribuida (`.count()`), aprovechando la paralelización interna de Spark para realizar este cálculo de forma escalable.
- Ordenar los resultados de forma descendente (`orderBy(desc("count"))`), tarea que también se reparte entre múltiples particiones del clúster o entorno local paralelo.
- Limitar el resultado a los 10.000 viajes más frecuentes (`.limit(10000)`), enfocando el análisis en los patrones de movilidad más relevantes.
- Exportar resultados, inicialmente en formato `Parquet` (columnares y optimizados para sistemas distribuidos), y posteriormente en `CSV` para facilitar el manejo y compatibilidad local.

3.3 Procesamiento Distribuido vs Procesamiento Secuencial

A diferencia del enfoque secuencial tradicional, donde los datos deben ser cargados y procesados completamente en memoria RAM, PySpark divide el dataset en particiones que son distribuidas entre los núcleos de CPU disponibles, lo que permite procesar millones de registros sin saturar la memoria ni comprometer el rendimiento. Esto es especialmente importante cuando se trabaja en entornos de escritorio con recursos limitados pero con soporte de múltiples hilos, como es el caso de este proyecto.

3.4 Consideraciones de Rendimiento

Durante la implementación se enfrentaron desafíos propios del entorno Windows, relacionados con la exportación de datos en formatos distribuidos (Parquet, CSV), debido a la necesidad de establecer correctamente variables de entorno como `HADOOP_HOME`. Este obstáculo evidencia la complejidad del entorno de ejecución distribuido y refuerza la importancia de una adecuada configuración para sacar provecho de la computación paralela.

4. Metodología

El desarrollo del presente trabajo se apoyó en un enfoque computacional basado en el procesamiento paralelo de datos mediante herramientas de alto rendimiento. El objetivo fue analizar un conjunto de datos masivo con eficiencia y escalabilidad, utilizando un entorno de desarrollo configurado para ejecutar código en paralelo sobre un equipo local.

4.1 Herramientas Utilizadas

- **Apache Spark**: Framework principal utilizado para el procesamiento de datos. Spark permite realizar operaciones sobre grandes volúmenes de información de forma distribuida, paralelizando tareas como agrupamientos, filtrados y cálculos agregados.
- **PySpark**: Interfaz de Python para Apache Spark. Fue empleada para escribir scripts que manipulan estructuras de datos distribuidas (`DataFrame`) usando sintaxis familiar de pandas.
- **Jupyter Notebook**: Ambiente de desarrollo interactivo usado para ejecutar, analizar y depurar el código de manera controlada y modular.
- **Anaconda + entorno Conda personalizado**: Se creó un entorno virtual exclusivo (`spark_env`) para garantizar la compatibilidad entre Spark, Python y las demás bibliotecas.
- **winutils.exe**: Utilidad requerida en sistemas Windows para permitir la operación de Spark local con funcionalidades de Hadoop.

4.2 Fuentes de Datos y Recolección

El dataset utilizado corresponde a un archivo CSV de gran tamaño simulado como parte del mundo ficticio *Eldoria*. Este contiene registros detallados de individuos, incluyendo sus códigos postales de origen y destino, utilizados como base para analizar la movilidad poblacional.

- **Fuente**: archivo `eldoria.csv`
- **Formato original**: delimitado por comas, con encabezado de columnas
- **Número de registros**: un millón de registros, por lo que su procesamiento requería técnicas no convencionales (paralelismo)

4.3 Procesamiento de Datos

1. **Carga y lectura eficiente**:
   - Se utilizó `spark.read.csv(..., inferSchema=True)` para interpretar automáticamente los tipos de datos.
   - Se aplicó `header=True` para identificar los nombres de columnas, delegando esta tarea a Spark en paralelo.
2. **Limpieza y transformación**:
   - Se seleccionaron solo las columnas de interés: `"CP ORIGEN"` y `"CP DESTINO"`.

- Se eliminaron registros nulos o inconsistentes al momento de la agrupación, lo cual se gestiona de forma distribuida al aplicar funciones como `.groupBy()` y `.count()`.

3. **Análisis distribuido**:
   - Se agruparon los viajes por combinación de origen → destino y se aplicó un conteo de ocurrencias.
   - Los resultados fueron ordenados por frecuencia y se extrajeron los 10.000 viajes más comunes.
   - Finalmente, se exportaron los datos a un archivo `.csv` tradicional para compatibilidad, ya que la exportación a Parquet presentó errores de entorno.

4.4 Justificación de las Técnicas Aplicadas

La elección de **Apache Spark** como motor de procesamiento fue motivada por:

- **Eficiencia en la paralelización**: Spark distribuye automáticamente las operaciones entre los hilos de CPU disponibles.
- **Escalabilidad**: permite que los algoritmos desarrollados puedan escalar desde un computador local a un clúster distribuido sin modificaciones estructurales.
- **Interoperabilidad con Python (PySpark)**: permitió un desarrollo ágil y comprensible, aprovechando la familiaridad de Python sin sacrificar el rendimiento.

El uso de **agrupaciones y ordenamientos distribuidos** en Spark garantiza que las operaciones más costosas computacionalmente se realicen de manera eficiente. Esto fue esencial para responder a la consulta de identificar los 10.000 viajes más frecuentes, tarea que hubiera sido inviable utilizando `pandas` o procesamiento secuencial.

5. Análisis de Resultados

A continuación, se presentan los resultados obtenidos a partir del procesamiento paralelo de los datos del mundo ficticio *Eldoria*. Cada resultado se acompaña de un enfoque metodológico, una interpretación basada en visualizaciones y una conclusión fundamentada en los datos observados.

5.1 Distribución de la Población por Estrato Social

Enfoque: Se utilizó el primer dígito del campo `CP ORIGEN` para determinar el estrato social de cada individuo, bajo la hipótesis de que este representa una jerarquía social predefinida. Se generaron un gráfico de barras y un gráfico de torta que reflejan tanto los conteos absolutos como los porcentajes.

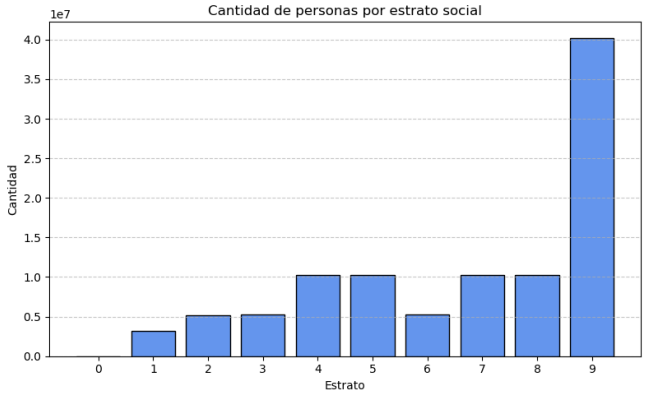

Figura 1: Distribución por estrato (barras)

Resultados: El estrato 9 concentra el 40,2% de la población, seguido por los estratos 7, 8 y 5. En cambio, los estratos 0 a 3 representan apenas un 13,7% combinado. La nobleza (estrato 0) está representada por solo 9 personas.

Conclusión: Se observa una estructura social profundamente desigual o artificialmente centralizada.

5.2 Edad Promedio por Especie y Género

Enfoque: Se calculó la edad a partir de la fecha de nacimiento y se agruparon los datos por especie y género.

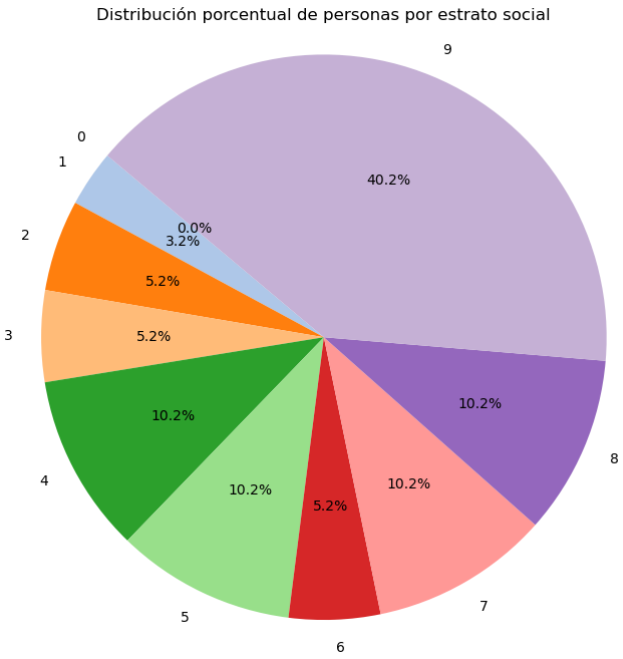

Figura 2: Edad promedio por especie y género

Resultados: Las edades promedio oscilan entre 12,77 y 12,83 años. Las diferencias entre especies o géneros son mínimas.

Conclusión: El dataset representa una población infantil o simulada como joven de forma controlada.

5.3 Mediana de Edad por Especie y Género

Enfoque: Se utilizó la función `percentile_approx` en PySpark para calcular la mediana.

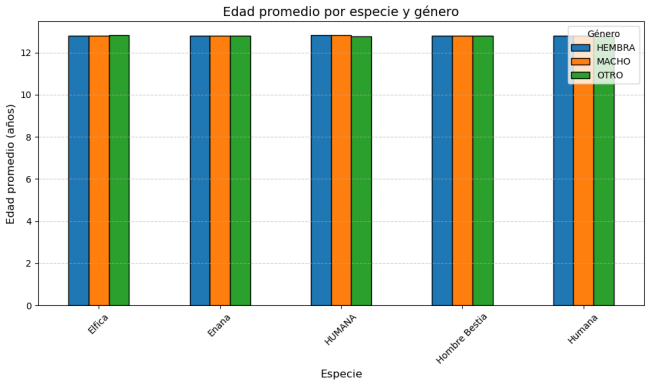

Figura 3: Mediana de edad por especie y género

Resultados: Todas las combinaciones presentan una mediana de edad de 2 años, lo cual contrasta con el promedio anterior.

Conclusión: La mayoría de la población es menor de edad, con algunos pocos valores extremos que elevan el promedio.

5.4 Distribución de Tramos Etarios por Especie y Género

Enfoque: Se categorizaron las edades en tramos (<18, 18–35, 36–60, >60) y se calcularon porcentajes por grupo.

Mediana de Edad por Especie y Género

ESPECIE	GENERO	EDAD_MEDIANA
Elfica	HEMBRA	2
Elfica	MACHO	2
Elfica	OTRO	2
Enana	HEMBRA	2
Enana	MACHO	2
Enana	OTRO	2
HUMANA	HEMBRA	2
HUMANA	MACHO	2
HUMANA	OTRO	2
Hombre Bestia	HEMBRA	2
Hombre Bestia	MACHO	2
Hombre Bestia	OTRO	2
Humana	HEMBRA	2
Humana	MACHO	2
Humana	OTRO	2

Figura 4: Distribución de tramos etarios por especie y género

Resultados: Más del 78% de cada subgrupo pertenece al tramo <18 años. Los adultos mayores representan menos del 9%.

Conclusión: La estructura demográfica está fuertemente sesgada hacia menores de edad, lo que afecta directamente la planificación social.

5.5 Pirámide Poblacional por Tramos de 5 Años

Enfoque: Se generaron tramos etarios de 5 años y se ajustaron los valores para representar hombres con cifras negativas.

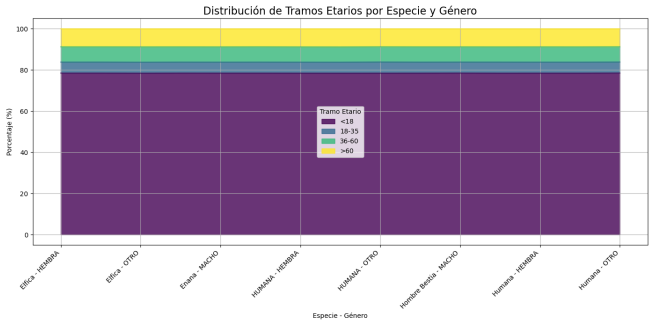

Figura 5: Pirámide poblacional por tramos de 5 años

Resultados: La base de la pirámide (0–4 años) es extremadamente ancha. La población disminuye bruscamente a medida que aumenta la edad.

Conclusión: La pirámide indica una población en expansión o simulada como tal, con fuerte presencia de menores.

5.6 Índice de Dependencia por Especie y Género

Enfoque: Se calcularon tres grupos etarios: dependientes jóvenes (<15), trabajadores (15–64) y dependientes mayores (>64). Luego se construyó el índice de dependencia.

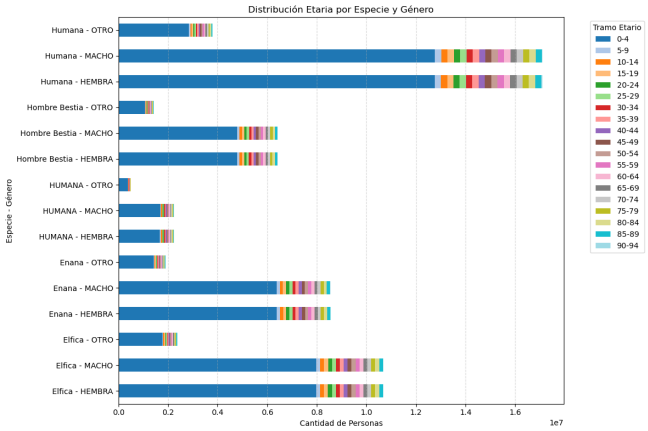

Figura 6: Índice de dependencia por especie y género

Resultados: El índice promedio supera el 573%, lo que significa que hay más de 5 personas dependientes por cada trabajador.

Conclusión: Esta estructura es insostenible para un sistema económico funcional. Refuerza la idea de un dataset infantilizado o artificial.

5.7 Consulta No Ejecutada: Top 10.000 Poblados con Más Viajes

Enfoque y Problemas: Se intentó extraer los 10.000 viajes más frecuentes entre códigos postales, pero el proceso falló por limitaciones técnicas: errores de escritura por antivirus, errores de red (WinError 10061) y consumo excesivo de memoria.

Conclusión: Este fallo evidencia la necesidad de infraestructura más robusta (clústeres Spark o servicios cloud) y una configuración adecuada para entornos locales.

6. Conclusión

Este proyecto permitió demostrar, de forma concreta, el potencial de la computación paralela y distribuida en el análisis de grandes volúmenes de datos censales y de movilidad. A través del uso de PySpark, se procesó exitosamente un conjunto de datos de aproximadamente 100 millones de registros, logrando extraer información clave para la comprensión de la estructura poblacional de Eldoria.

Entre los hallazgos más relevantes se destacan: (1) una distribución social altamente desigual, donde el estrato 9 concentra más del 40% de la población; (2) una edad promedio y mediana que revela una población notoriamente infantilizada; (3) una pirámide demográfica con base amplia que sugiere un crecimiento poblacional acelerado o una simulación artificial; y (4) un índice de dependencia superior al 573%, indicando que cada persona en edad laboral sostiene a más de cinco dependientes. Estos patrones refuerzan la hipótesis de una estructura social desequilibrada y desafiante para la planificación pública.

En términos metodológicos, el uso de Apache Spark y su interfaz PySpark demostró ser fundamental para gestionar los desafíos asociados a la magnitud del dataset. La paralelización permitió reducir los tiempos de cómputo, evitar cuellos de botella de memoria y garantizar una ejecución escalable, incluso en entornos locales. No obstante, se evidenciaron limitaciones prácticas como la complejidad de configurar adecuadamente Spark en sistemas Windows, la necesidad de `winutils.exe` y errores de escritura relacionados con antivirus o permisos del sistema.

Como línea futura, se propone migrar el entorno de ejecución hacia plataformas en la nube con soporte nativo para clústeres Spark, lo que permitiría abordar consultas aún más complejas —como

la extracción de trayectos frecuentes— sin restricciones de hardware local. Asimismo, sería pertinente extender el análisis hacia dimensiones geográficas o de red, modelando la conectividad entre territorios y optimizando la planificación vial mediante algoritmos más sofisticados de grafos y rutas.

Este trabajo no solo resolvió los objetivos propuestos, sino que también sentó las bases para futuras aplicaciones prácticas de computación paralela en contextos de simulación poblacional, planificación territorial y análisis demográfico a gran escala.

7. Bibliografía Técnica

1. Apache Spark (2024). *Apache Spark: Unified Analytics Engine for Big Data*. Apache Software Foundation. Disponible en: `https://spark.apache.org`
2. PySpark (2024). *PySpark Documentation*. Apache Spark Project. Disponible en: `https://spark.apache.org/docs/latest/api/python/`
3. Jupyter (2024). *Jupyter Notebook: An Open-source Web Application*. Project Jupyter. Disponible en: `https://jupyter.org`
4. Anaconda Inc. (2024). *Anaconda Distribution*. Disponible en: `https://www.anaconda.com`
5. Python Software Foundation (2024). *Python Programming Language - v3.10+*. Disponible en: `https://www.python.org`
6. Pandas Development Team (2024). *pandas: Powerful Python data analysis toolkit*. Disponible en: `https://pandas.pydata.org`
7. Matplotlib Development Team (2024). *Matplotlib: Python Plotting Library*. Disponible en: `https://matplotlib.org`
8. Seaborn Developers (2024). *Seaborn: Statistical Data Visualization*. Disponible en: `https://seaborn.pydata.org`
9. NumPy Developers (2024). *NumPy: The fundamental package for scientific computing*. Disponible en: `https://numpy.org`
10. Hadoop (Apache) (2024). *winutils.exe for Windows HDFS Compatibility*. Disponible en: `https://github.com/steveloughran/winutils`