

Informe sobre el Sistema de Búsqueda y Hadoop

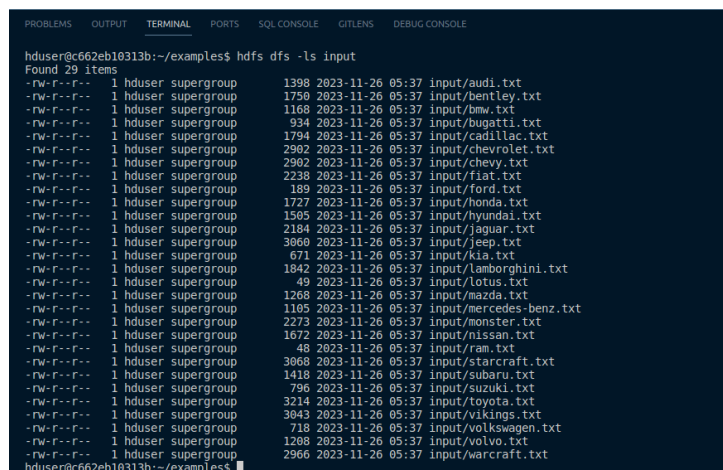
Nicolas Rojas

November 26, 2023

1 Introducción

El procesamiento eficiente de grandes volúmenes de datos se ha convertido en una necesidad crucial en la era digital actual. En este contexto, Hadoop emerge como una herramienta poderosa y versátil que aborda los desafíos asociados con el manejo de conjuntos de datos masivos. Desarrollado como un proyecto de código abierto, Hadoop proporciona una plataforma robusta que permite el procesamiento distribuido y paralelo de datos a través de clusters de hardware convencional. Su arquitectura escalable y su capacidad para tolerar fallos lo convierten en una opción destacada para empresas e instituciones que buscan extraer valor significativo de sus enormes depósitos de información.

Hadoop consta de varios componentes clave, como el Sistema de Archivos Distribuido HDFS (Hadoop Distributed File System) y el paradigma de procesamiento distribuido MapReduce. Estos elementos trabajan en conjunto para almacenar y procesar grandes cantidades de datos de manera eficiente. Además, Hadoop proporciona flexibilidad al integrarse con una variedad de herramientas y frameworks que amplían sus capacidades, permitiendo el análisis, la gestión y la extracción de información valiosa de conjuntos de datos a una escala que antes parecía inabordable.



```
hdfsuser@cc662eb10313b:~/examples$ hdfs dfs -ls input
Found 29 items
-rw-r--r-- 1 hdfsuser supergroup 1398 2023-11-26 05:37 input/audi.txt
-rw-r--r-- 1 hdfsuser supergroup 1750 2023-11-26 05:37 input/bentley.txt
-rw-r--r-- 1 hdfsuser supergroup 1168 2023-11-26 05:37 input/bmw.txt
-rw-r--r-- 1 hdfsuser supergroup 934 2023-11-26 05:37 input/bugatti.txt
-rw-r--r-- 1 hdfsuser supergroup 1794 2023-11-26 05:37 input/cadillac.txt
-rw-r--r-- 1 hdfsuser supergroup 2902 2023-11-26 05:37 input/chevrolet.txt
-rw-r--r-- 1 hdfsuser supergroup 2902 2023-11-26 05:37 input/chevy.txt
-rw-r--r-- 1 hdfsuser supergroup 2238 2023-11-26 05:37 input/fiat.txt
-rw-r--r-- 1 hdfsuser supergroup 189 2023-11-26 05:37 input/ford.txt
-rw-r--r-- 1 hdfsuser supergroup 1727 2023-11-26 05:37 input/honda.txt
-rw-r--r-- 1 hdfsuser supergroup 1505 2023-11-26 05:37 input/hyundai.txt
-rw-r--r-- 1 hdfsuser supergroup 2184 2023-11-26 05:37 input/jaguar.txt
-rw-r--r-- 1 hdfsuser supergroup 3060 2023-11-26 05:37 input/jeep.txt
-rw-r--r-- 1 hdfsuser supergroup 671 2023-11-26 05:37 input/kia.txt
-rw-r--r-- 1 hdfsuser supergroup 1842 2023-11-26 05:37 input/lamborghini.txt
-rw-r--r-- 1 hdfsuser supergroup 49 2023-11-26 05:37 input/lotus.txt
-rw-r--r-- 1 hdfsuser supergroup 1268 2023-11-26 05:37 input/mazda.txt
-rw-r--r-- 1 hdfsuser supergroup 1105 2023-11-26 05:37 input/mercedes-benz.txt
-rw-r--r-- 1 hdfsuser supergroup 2273 2023-11-26 05:37 input/monster.txt
-rw-r--r-- 1 hdfsuser supergroup 1672 2023-11-26 05:37 input/nissan.txt
-rw-r--r-- 1 hdfsuser supergroup 48 2023-11-26 05:37 input/ram.txt
-rw-r--r-- 1 hdfsuser supergroup 3068 2023-11-26 05:37 input/starcraft.txt
-rw-r--r-- 1 hdfsuser supergroup 1418 2023-11-26 05:37 input/subaru.txt
-rw-r--r-- 1 hdfsuser supergroup 796 2023-11-26 05:37 input/suzuki.txt
-rw-r--r-- 1 hdfsuser supergroup 3214 2023-11-26 05:37 input/toyota.txt
-rw-r--r-- 1 hdfsuser supergroup 3043 2023-11-26 05:37 input/vikings.txt
-rw-r--r-- 1 hdfsuser supergroup 718 2023-11-26 05:37 input/volkswagen.txt
-rw-r--r-- 1 hdfsuser supergroup 1208 2023-11-26 05:37 input/volvo.txt
-rw-r--r-- 1 hdfsuser supergroup 2966 2023-11-26 05:37 input/warcraft.txt
hdfsuser@cc662eb10313b:~/examples$
```

Figure 1: Se completa la creacion de carpetas con los nombres dados

2 Explicación de Solución: Wordcount

2.1 Obtención de Documentos

Se realizó una solicitud a la API de Wikipedia para obtener información sobre una lista de entradas relacionadas con automóviles y otros términos. Cada documento se almacenó en archivos de texto en dos carpetas distintas según su posición en la lista.

Word	[(Document1, Count1), ...]
Hola	(1, 2) (2, 2)
soy	(1, 1) (2, 2)
y	(1, 1)
...

Figure 2: Resultado Wordcount

2.2 Mapper (mapper.py)

El script ‘mapper.py’ procesa cada documento obtenido de la API. Convierte el texto a minúsculas, elimina caracteres especiales y divide la entrada en nombre del documento y contenido. Genera pares clave-valor para cada palabra en el documento en el formato ‘palabra_tnombre_documento_t1’.

2.3 Shuffling y Sorting

La salida del mapper se envía a la fase de shuffling y sorting, donde los datos se agrupan y ordenan según las claves.

2.4 Reducer (reducer.py)

El script ‘reducer.py’ procesa la salida ordenada de los mappers. Agrega las cuentas de las palabras por documento en una estructura de datos y genera la salida final en el formato ‘palabra[(Documento1, Conteo1), ...]’.

3 Explicación de Solución: Wordcount Inverted Index

La solución WordCount Inverted Index extiende el enfoque básico de WordCount mediante la implementación de un índice invertido. Un índice invertido es una estructura de datos clave en los motores de búsqueda y facilita la búsqueda rápida de información. A diferencia del enfoque tradicional de WordCount que simplemente cuenta la frecuencia de palabras en cada documento, el índice invertido permite consultar qué documentos contienen una palabra específica y cuántas veces aparece en cada uno.

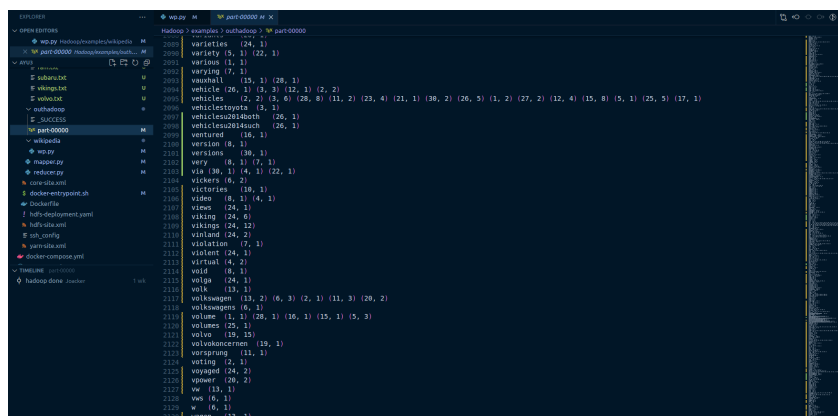


Figure 3: Resultado Index inverted

3.1 Mapper (`_textttmapper_inverted.py`)

El script `mapper_inverted.py` sigue un proceso similar al WordCount básico, realizando las siguientes acciones:

- Convierte el texto a minúsculas.
- Elimina caracteres especiales.
- Divide la entrada en el nombre del documento y su contenido.
- Genera pares clave-valor para cada palabra en el documento en el formato `_textttpalabra_t nombre_documentoí`.

3.2 Shuffling y Sorting

La fase de shuffling y sorting agrupa y ordena los datos según las claves, preparándolos para la fase de reducción.

3.3 Reducer (`_textttreducer_inverted.py`)

El script `_textttreducer_inverted.py` procesa la salida ordenada de los mappers y realiza las siguientes tareas:

- Construye un índice invertido que mapea cada palabra a la lista de documentos en los que aparece y la frecuencia en cada documento.
- Imprime la salida final en el formato `palabraí (Documento1, Conteo1), ... í`.

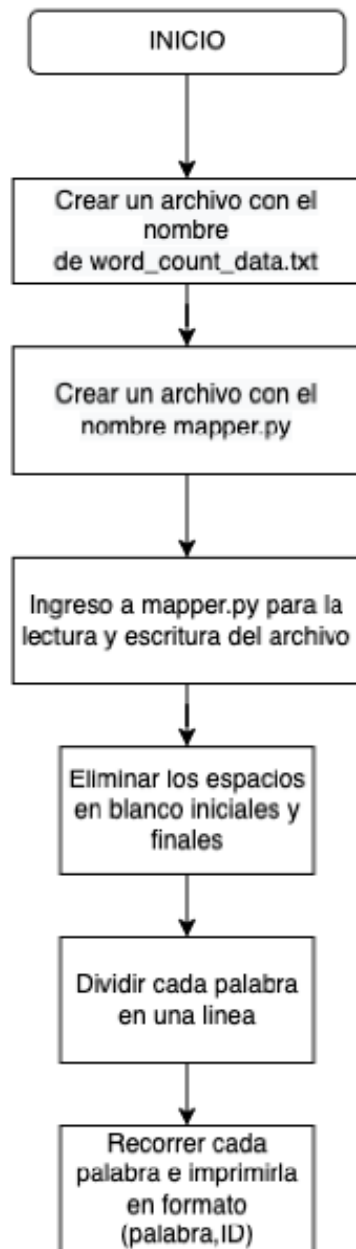


Figure 4: Diagrama de funcionamiento Hadoop

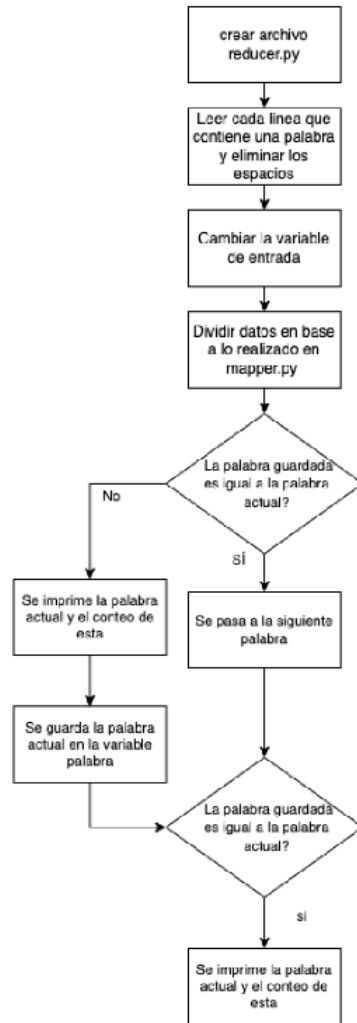


Figure 5: Diagrama de funcionamiento Index invertido

```

1 import json
2
3 input_file_path = "part-00000"
4 output_file_path = "/alacranes/datos.json" # Puedes cambiar el nombre del archivo segun tus preferencias
5
6 data_list = [] # Lista para almacenar los datos
7
8 with open(input_file_path, "r") as input_file:
9     for line in input_file:
10         # Separa la palabra y la lista de documentos y frecuencias
11         word, doc_list_str = line.strip().split("\t")
12         # Elimina los corchetes y espacios y divide la lista en pares (documento, frecuencia)
13         doc_list = [tuple(map(str.strip, pair.split("(").split(")"))) for pair in doc_list_str.split(" ")]
14
15         # Crea un diccionario con la informacion
16         data = {"word": word, "documents": doc_list}
17         # Agrega el diccionario a la lista
18         data_list.append(data)
19
20 # Escribe la lista de datos en un archivo JSON
21 with open(output_file_path, "w") as output_file:
22     json.dump(data_list, output_file, indent=2)
23
24 print(f"Los datos se han almacenado en {output_file_path}")

```

Figure 6: Código para generar un JSON donde se almacene la información rescatada

```

1 import json
2
3 class Buscador:
4     def __init__(self, datos):
5         self.datos = datos
6
7     def buscar(self, consulta):
8         consulta = consulta.lower()
9
10        # Busca la palabra en los datos
11        # Retorna una lista de documentos y frecuencias
12        resultados = []
13
14        # Obtener los documentos ordenados por coincidencias (de mayor a menor)
15        documentos_ordenados = sorted(self.datos[consulta], key=lambda x: len(x[1]), reverse=True)
16
17        # Obtener los 5 mejores resultados
18        for doc_id, coincidencias in documentos_ordenados[:5]:
19            url = f"http://es.wikipedia.org/wiki/{doc_id}"
20            resultados.append((url, doc_id, len(coincidencias)))
21
22        result = "Resultados para '{consulta}':\n"
23        for url, doc_id, frecuencia in resultados:
24            result += f"Documento: {doc_id}, Frecuencia: {frecuencia}, URL: {url}\n"
25
26        return result
27
28 if __name__ == "__main__":
29     # Leer los datos desde el archivo JSON
30     with open("alacranes/datos.json", "r") as f:
31         datos = json.load(f)
32
33     buscador = Buscador(datos)
34
35     # Buscar
36     consulta = input("Ingresa tu búsqueda (o presiona enter para salir): ")
37     resultado = buscador.buscar(consulta)
38     print(resultado)
39
40     # Salir
41     if consulta == "":
42         print("Ingresa tu búsqueda (o presiona enter para salir): ")

```

Figure 7: Código del buscador

4 Preguntas

1. Alternativas a HDFS y características distintivas.

- Ceph: Proporciona un sistema de archivos distribuido altamente escalable y tolerante a fallos.
- GlusterFS: Un sistema de archivos distribuido que puede ser utilizado como alternativa a HDFS.
- Amazon S3 (Simple Storage Service): Aunque no es un sistema de archivos distribuido tradicional, es utilizado ampliamente para almacenamiento distribuido en la nube.

Características diferenciadoras de HDFS:

- Arquitectura de Replicación: HDFS utiliza la replicación para garantizar la tolerancia a fallos y la disponibilidad de los datos.
- Optimizado para Grandes Volúmenes: HDFS está diseñado específicamente para manejar grandes volúmenes de datos, siendo eficiente para tareas de procesamiento masivo.

2. Tolerancia a fallos y recuperación de datos en Hadoop.

- Tolerancia a Fallos en HDFS: HDFS logra la tolerancia a fallos mediante la replicación de bloques de datos en varios nodos del clúster. Si un nodo falla, los bloques replicados en otros nodos están disponibles.
- Recuperación de Datos en MapReduce: Si una tarea Map o Reduce falla, se reinicia automáticamente en otro nodo. Además, si un nodo de trabajo falla, el trabajo completo se redistribuye a otros nodos.

3. Caso real de uso de Apache Hadoop.

- Yahoo!: Implementó Hadoop para analizar grandes conjuntos de datos web y mejorar la eficiencia de su motor de búsqueda. Hadoop permitió procesar datos a escala, mejorando la calidad de los resultados de búsqueda.
4. Otros frameworks comúnmente utilizados con Hadoop.
- Apache Spark: Utilizado para procesamiento de datos en memoria y análisis de datos en tiempo real.
 - Apache Flink: Un sistema de procesamiento de datos en tiempo real y procesamiento de flujos.
 - Apache Storm: Enfoque en procesamiento de eventos en tiempo real.
5. Lenguajes de consulta de alto nivel en Hadoop MapReduce.
- HiveQL (Hive Query Language): Similar a SQL, HiveQL facilita la consulta de datos almacenados en Hadoop utilizando un lenguaje más familiar.
 - Pig Latin: Un lenguaje de scripting utilizado con Apache Pig, que simplifica el desarrollo de tareas en MapReduce.

5 Código del Generador de Documentos

```
import requests
import json

# Lista de entradas (30 documentos)
entradas = [
    "mazda", "nissan", "toyota", "warcraft", "suzuki",
    "bentley", "monster", "starcraft", "lotus", "bugatti",
    "audi", "mercedes-benz", "volkswagen", "ford", "chevrolet",
    "honda", "hyundai", "subaru", "volvo", "lamborghini",
    "kia", "jaguar", "bmw", "vikings", "fiat",
    "jeep", "cadillac", "chevy", "ram", "subaru"
]
url = "https://en.wikipedia.org/w/api.php"
i = 1
for entrada in entradas:
    params = {
        'format': 'json',
        'action': 'query',
        'prop': 'extracts',
        'exintro': '',
        'explaintext': '',
        'redirects': 1,
        'titles': entrada
    }

    req = requests.get(
        url,
        params=params
    ).json()

    n_page = list(req['query']['pages'].keys())[0]
    texto = req['query']['pages'][n_page]['extract']
    texto = '{}<splittername>{}'.format(i, json.dumps(texto))

    if i <= 15:
        with open(f'../carpeta1/{entrada}.txt', 'w') as f:
            f.write(texto)
    else:
        with open(f'../carpeta2/{entrada}.txt', 'w') as f:
            f.write(texto)
    i = i + 1
```

6 Conclusiones

En conclusión, el uso de Hadoop marca un hito significativo en la gestión de datos a gran escala. Su capacidad para procesar información de manera distribuida, aprovechando la potencia de clusters de hardware convencional, ha revolucionado la forma en que las organizaciones abordan el análisis y procesamiento de grandes volúmenes de datos. La escalabilidad, tolerancia a fallos y su integración con diversos componentes del ecosistema Big Data hacen de Hadoop una opción preferida para empresas que buscan gestionar, analizar y comprender sus enormes cantidades de datos de manera efectiva. En un mundo donde los datos son el activo más valioso, Hadoop se destaca como una herramienta esencial para convertir la complejidad de los datos masivos en conocimiento valioso.

7 Video Demostrativo

<https://github.com/Delaphont/Sist-distribuidos/tree/main/tarea3>