UNIVERSIDAD DIEGO PORTALES

FACULTAD DE INGENIERÍA Y CIENCIAS
ESCUELA DE INFORMÁTICA Y TELECOMUNICACIONES

Sistemas Distribuidos
Tarea 3

Profesor:
Nicolás Hidalgo

Ayudantes:
Nicolás Nuñez
Cristian Villavivencio
Joaquin Fernandez

Estudiantes:
Nicolás Moncada
Dayana Ruminot

Santiago, Chile **27 de Noviembre del 2022**

Índice

1. Problema y Solución

Los buscadores implementados dentro del curso no son completamente eficientes. Se han sacado datos de distintas páginas y han sido guardados directamente en la base de datos. Sin embargo, la técnica utilizada es lenta y redundante¿Para qué tener datos repetidos (los documentos de texto y los datos en la base de datos)? ¿Hay una forma más eficiente? Dicho esto, se propone armar un buscado que funcione de mejor manera a modo de desafío personal, utilizando MapReduce.

Para dar solución a este problema es realizar lo siguiente:

- Obtener 10 documentos de texto utilizando la API de wikipedia. Para este ítem es necesario mostrar el código empleado.
- Dividir los 10 documentos del ítem anterior en dos carpetas distintas. Desde el documento 1 al 5, irían en la carpeta 1. Desde el documento 6 al 10 irían en la carpeta 2.
- Investigar un código de MapReduce que permita contar palabras en un documento. En caso de ser necesario se solicita mostrar el código. Adicionalmente, se debe mostrar un diagrama que explique el funcionamiento del código.
- Generar un trabajo de WordCount para todos los documentos, utilizando Hadoop. Este trabajo de WordCount generará Key-values. Por ejemplo:
  - Documento1: Hola soy Nicolás y él es Joaquín.
  - Documento2: Hola soy Nicolás. Hola soy Joaquín.

  El output del trabajo de WordCount generar a los siguientes valores:

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

Figura 1: Output del trabajo de WordCount.

- Guardar los valores anteriores en alguna base de datos. Puede ser de cualquier tipo.
- Generar un buscador el cual deberá utilizar como regla el buscar la mayor cantidad de coincidencias posibles utilizando el índice invertido. Finalmente, deberá retornar URLs que permitan ver los documentos de manera específica.
- Como último paso se debe documentar en vídeo de no más de 10 minutos todo el funcionamiento del buscador.

2. Códigos

Todos los códigos se pueden encontrar en el repositorio `https://github.com/NicoMonc/Tarea3SD`

2.1. Extracción de 10 documentos de texto y división de documentos en carpetas

Se solicita extraer 10 documentos de Wikipedia, para posteriormente dividir estos 10 documentos en 2 grupos paralelos, para realizar esta tarea inicial se implementó el siguiente código:

```python
import urllib.request
import os

for i in range(10):
    imprimir=i+1
    opcion = input ("Texto a buscar: ")
    print(opcion)
    if(i<5):
        site = urllib.request.urlopen('https://es.wikipedia.org/w/api.php?action=query&list=search&srprop=snippet&format=json&origin=*&utf8=&srsearch='+opcion)
        data = site.read()
        print(data)

        isFile = os.path.isfile("./1_5/search"+str(imprimir)+".txt")
        if (isFile):
            os.remove("./1_5/search"+str(imprimir)+".txt")

        crear = open("./1_5/search"+str(imprimir)+".txt","wb") #open file in binary mode
        crear.write(data)
        crear.close()
    else:
        site = urllib.request.urlopen('https://es.wikipedia.org/w/api.php?action=query&list=search&srprop=snippet&format=json&origin=*&utf8=&srsearch='+opcion)
        data = site.read()
        print(data)

        isFile = os.path.isfile("./6_10/search"+str(imprimir)+".txt")
        if (isFile):
            os.remove("./6_10/search"+str(imprimir)+".txt")

        crear = open("./6_10/search"+str(imprimir)+".txt","wb") #open file in binary mode
        crear.write(data)
        crear.close()
```

Figura 2: API.py

En el código anterior se puede ver como mediante la api se accede al sitio web solicitado en la tarea, para posteriormente imprimir y separar en cada una de las carpetas solicitadas, integrando en la primera carpeta los primeros 5 archivos de texto y luego los 5 restantes en la segunda carpeta.

Previamente a separar los datos se recorren con mappereduce y reducer como se verá a continuación.

2.2. WordCount para cada documento

Esta sección se realiza con mapreduce.py y reducer.py, siendo el primero una revisión de cada línea, para posteriormente en reducer realizar el recuento de palabras por cada archivo. Por lo cual, lo primero que se realiza posteriormente a el WordCount de los 10 archivos es guardarlos directamente en la base de datos, para posteriormente ser utilizado en api.py

```python
import sys

for line in sys.stdin:
    line = line.strip()
    words = line.split()

    for word in words:
        print ("%s\t%s" % (word, 1))
```

Figura 3: mapreduce.py

```python
from operator import itemgetter
import sys

current_word = None
current_count = 0
word = None

for line in sys.stdin:
    line = line.strip()
    word, count = line.split('\t', 1)
    try:
        count = int(count)
    except ValueError:
        continue

    if current_word == word:
        current_count += count
    else:
        if current_word:
            print ("%s\t%s" % (current_word, current_count))
        current_count = count
        current_word = word

if current_word == word:
    print ("%s\t%s" % (current_word, current_count))
```

Figura 4: reducer.py

2.3. Guardar datos en base de datos y buscador

Respecto a la base de datos, se tienen dos archivos para el funcionamiento, uno es bd.py y otra que trabaja con sql generando las tablas para guardar la información obtenida. El código de bd se ejecuta al previo funcionamiento de mapreduce.py y reducer.py con la intención de guardar en la base de datos sql los datos como; ID, palabra, número (wordcount), archivo.

Posteriormente atener los datos registrados se tiene el buscador que funciona dentro de la api, de momento no nos fue posible conectar nuestro proceso con hadoop. Sin embargo idea planteada es levantar una instancia en docker y realizar una conexión dentro de la terminal ingresar los comandos para conectar la api a hadoop como lo son HDFS y finalmente ingresar los comandos que están en el repositorio reverenciado (Hadoop Streaming Using Python).

3. Hadoop Streaming

El código de MapReduce elegido para contar las palabras de un documento es Hadoop Streaming. Hadoop Streaming es una característica que viene con Hadoop y permite a los usuarios o desarrolladores usar varios lenguajes diferentes para escribir programas de MapReduce como Python, C++, Ruby, etc. Es compatible con todos los lenguajes que pueden leer desde la entrada estándar y escribir en la salida estándar. Implementaremos Python con Hadoop Streaming y observaremos cómo funciona. Implementaremos el problema de conteo de palabras en python para comprender Hadoop Streaming.

Para realizar el mapeo y la reducción se utilizan mapper.py y reducer.py

- **mapper.py** toma un archivo de texto, lee el archivo, separa las palabras de acuerdo a los espacios que existan entre ellas, guarda la palabra en un arreglo, recorre la palabra y las imprime con un salto de linea entre cada una.

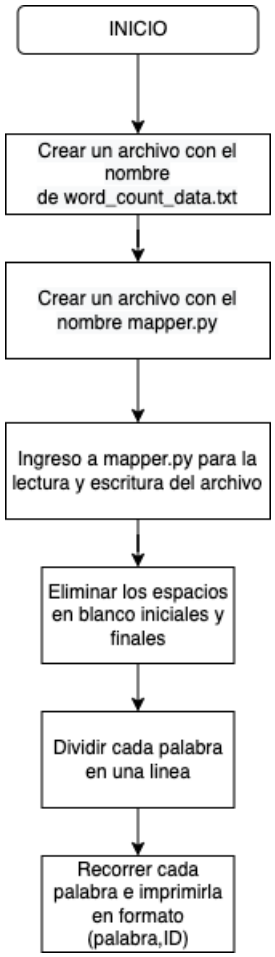

Figura 5: Diagrama del funcinamiento de mapper.py

- **reducer.py** Posteriormente a crear mapper.py se inicializan los demonios de hadoop, se crea una carpeta y dentro de este **crearword**$_{c}ount_{d}ata.txt$

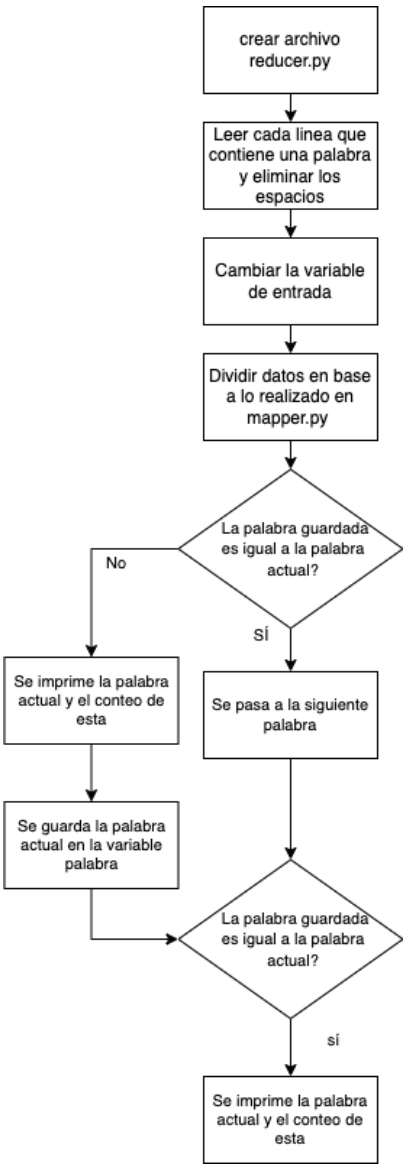

Figura 6: Diagrama del funcionamiento de reducer.py

Finalmente, el diagrama del funcionamiento en conjunto de ambos códigos y la configuración por terminal para autorizar lectura y escritura es el siguiente:

Figura 7: Diagrama de funcionamiento

4. Bibliografía

- **Vídeo del funcionamiento**

- **Repositorio**
  `https://github.com/NicoMonc/Tarea3SD`
- **Hadoop Streaming Using Python**
  `https://www.geeksforgeeks.org/hadoop-streaming-using-python-word-count-problem/`