

# Informe de la implementación de PageRank + Índice invertido en un cluster de hadoop

Ricardo Manuel Lazo Vásquez - Universidad Catolica San Pablo

<b>Introducción</b>	<b>1</b>
<b>El corpus de Google</b>	<b>1</b>
<b>Implementaciones</b>	<b>2</b>
Implementación lineal	2
Script allocator.py	2
Script content.py	2
Script pagerank.py	3
Script dic_generator.py	3
Script inverted_index.py	3
Implementación en el cluster de hadoop	3
Comparación	3
<b>Consideraciones para la ejecucion</b>	<b>7</b>
<b>Ejecución</b>	<b>7</b>
<b>Conclusiones</b>	<b>8</b>
<b>Referencias</b>	<b>8</b>

## Introducción

El presente informe busca detallar la implementación de un buscador utilizando índice invertido y PageRank sobre la base de datos del corpus de Google.

## El corpus de Google

Esta base de datos cuenta con 32 archivos con un peso acumulado de 5GB compuesta por pares de palabras y un puntaje. Para esta implementación solo se considero las filas de los archivos cuyos pares sean palabras y no contengan símbolos ni números, se ignoró el puntaje de repetición y mapeamos las columnas de la siguiente manera:

Columna	Uso en la implementación
Columna 1	Es utilizada como subpágina o referencia

	de la pagina "Filename"; cuando otro filename apunta a una palabra y esta es la columna 1 del archivo se considera una referencia del primer al segundo archivo. Ej: 2gm-0000 tiene una fila con la columna 1 la palabra "bio", bio esta en la columna 2 del archivo 2gm-0004 14 veces, entonces se considera que el archivo (o pagina) 2gm-0004 apunta 14 veces mas al archivo 2gm-0000.
Columna 2	Es considerado el puntero hacia otros archivos.
Columna 3 (Puntaje)	Dato descartado.
Columna 1 y Columna 2	Se consideran ambas columnas como parte del índice invertido, en este se cuentan las repeticiones/palabra asignando un puntaje en la carpeta Resultados.

## Implementaciones

### Implementación lineal

En la carpeta scripts se encuentran los scripts en python para la implementación lineal sin hadoop. Se implementó primero linealmente por temas de comprensión los archivos en orden de ejecución son:

1. allocator.py
2. content.py
3. pagerank.py
4. dic\_generator.py
5. inverted\_index.py

#### Script allocator.py

La función de allocator.py es ubicar que palabras pertenecen a cada archivo "2gm-00\*\*" o columna 1, cuantas lineas validas posee (o número total de referencias), y cuales son sus columnas 2 o punteros.

#### Script content.py

content.py genera un grafo de referencias de todos los archivos a todos los archivos, recorre los archivos generados por allocator para ese propósito.

### Script pagerank.py

pagerank.py es el unico archivo que no es tan necesario ejecutar con mapreduce, ejecuta la función PageRank recorriendo el grafo de content.py y utilizando el número total de referencias por archivo que producía allocator.py en un numero pequeño de iteraciones (Usamos it = 3) ya que al aumentar las iteraciones el PageRank de las páginas se vuelve 0.

### Script dic\_generator.py

Script que genera un diccionario global de todos los archivos produciendo un json de elementos clave valor "palabra": True, este script es necesario para ejecutar el índice invertido.

### Script inverted\_index.py

Genera por cada palabra en el diccionario anterior un json de puntajes por archivo, este script recorre todos los archivos del corpus por palabra por lo que su ejecución es lenta y altamente paralelizable.

## Implementación en el cluster de hadoop

Se aplicó la arquitectura mapreduce sobre todos los scripts menos pagerank.py, para ello se generó un grupo de carpetas de salida en la carpeta Resultados y se crearon scripts por cada etapa de mapreduce en la carpeta map-reduce\_scripts

## Comparación

allocator.py

```
1  import json
2  import re
3
4  def main():
5      r = "[a-z]+"
6      with open("filenames.json", "r") as fn:
7          allfn = json.load(fn)
8      # Map refs
9      print("iniciando mapeo de referencias")
10     # dic of page: location
11     dw = {}
12     # Number of refs per file
13     drefs = {}
14     # word and page Content
15     print("begin")
16     for k in allfn["filenames"]:
17         print("Processing file: " + k)
18         df = {}
19         with open("../Datos/" + k, "r") as tf:
20             drefs[k] = 0
21             while True:
22                 line = tf.readline()
23                 if not line:
24                     break
25                 line = line.lower()
26                 line = re.findall(r, line)
27                 if len(line) > 1:
28                     drefs[k] += 1
29                     dw[line[0]] = k
30                     df[line[1]] = True
31             with open("../Resultados/Content/" + k + "-content.json", "w") as js:
32                 json.dump(df, js)
33         with open("../Resultados/Content/refs.json", "w") as refs:
34             json.dump(drefs, refs)
35         with open("../Resultados/Content/pos.json", "w") as pos:
36             json.dump(dw, pos)
37
38     if __name__ == "__main__":
39         main()
```

map1.py y reduce1.py

```
1  #!/usr/bin/env python3
2
3  import json
4  import re
5  import sys
6
7  r = "[a-z]+"
8  for k in sys.stdin:
9      k = k[:-1]
10     dw = {}
11     # Number of refs per file
12     drefs = {}
13     # word and page Content
14     print("Processing file: " + k)
15     df = {}
16     with open("../Datos/" + k, "r") as tf:
17         drefs[k] = 0
18         while True:
19             line = tf.readline()
20             if not line:
21                 break
22             line = line.lower()
23             line = re.findall(r, line)
24             if len(line) > 1:
25                 drefs[k] += 1
26                 dw[line[0]] = k
27                 df[line[1]] = True
28         with open("../Resultados/map1/" + k + "-content.json", "w") as js:
29             json.dump(df, js)
30         with open("../Resultados/map1/" + k + "-refs.json", "w") as refs:
31             json.dump(drefs, refs)
32         with open("../Resultados/map1/" + k + "-pos.json", "w") as pos:
33             json.dump(dw, pos)
34
```

```
reduce1.py x
1  #!/usr/bin/env python3
2
3  import json
4  from otherfuns import *
5
6  def main():
7      with open("filenames.json", "r") as fn:
8          filenames = json.load(fn)
9          filenames = filenames["filenames"]
10         drefs = {}
11         dw = {}
12         for f in filenames:
13             print("[+] Working on " + f)
14             with open("../Resultados/map1/" + f + "-refs.json", "r") as trefs:
15                 tr = json.load(trefs)
16                 drefs = joinDictionaries(drefs, tr)
17             with open("../Resultados/map1/" + f + "-pos.json", "r") as tpos:
18                 tp = json.load(tpos)
19                 dw = joinDictionaries(dw, tp)
20             with open("../Resultados/reduce1/refs.json", "w") as refs:
21                 json.dump(drefs, refs)
22             with open("../Resultados/reduce1/pos.json", "w") as pos:
23                 json.dump(dw, pos)
24             print("[+] Done!")
25
26 if __name__ == "__main__":
27     main()
28
```

Y de esta manera para todos los scripts, todo archivo unico que salia de cada etapa se divide en archivos por archivo "2gm-00\*\*". En el caso del script de índice invertido se ejecutó con las palabras del diccionario.

Para la ejecución en mapreduce se debe ejecutar el streaming de hadoop con los archivos a ejecutar, los archivos de entrada y la carpeta de salida, el comando general sería el siguiente:

```
hadoop jar /pat/a//hadoop/tools/lib/hadoop-streaming-3.2.1.jar \
-file map.py -mapper map.py \
-file reduce.py -reducer reduce.py \
-input input.txt -output output_dir
```

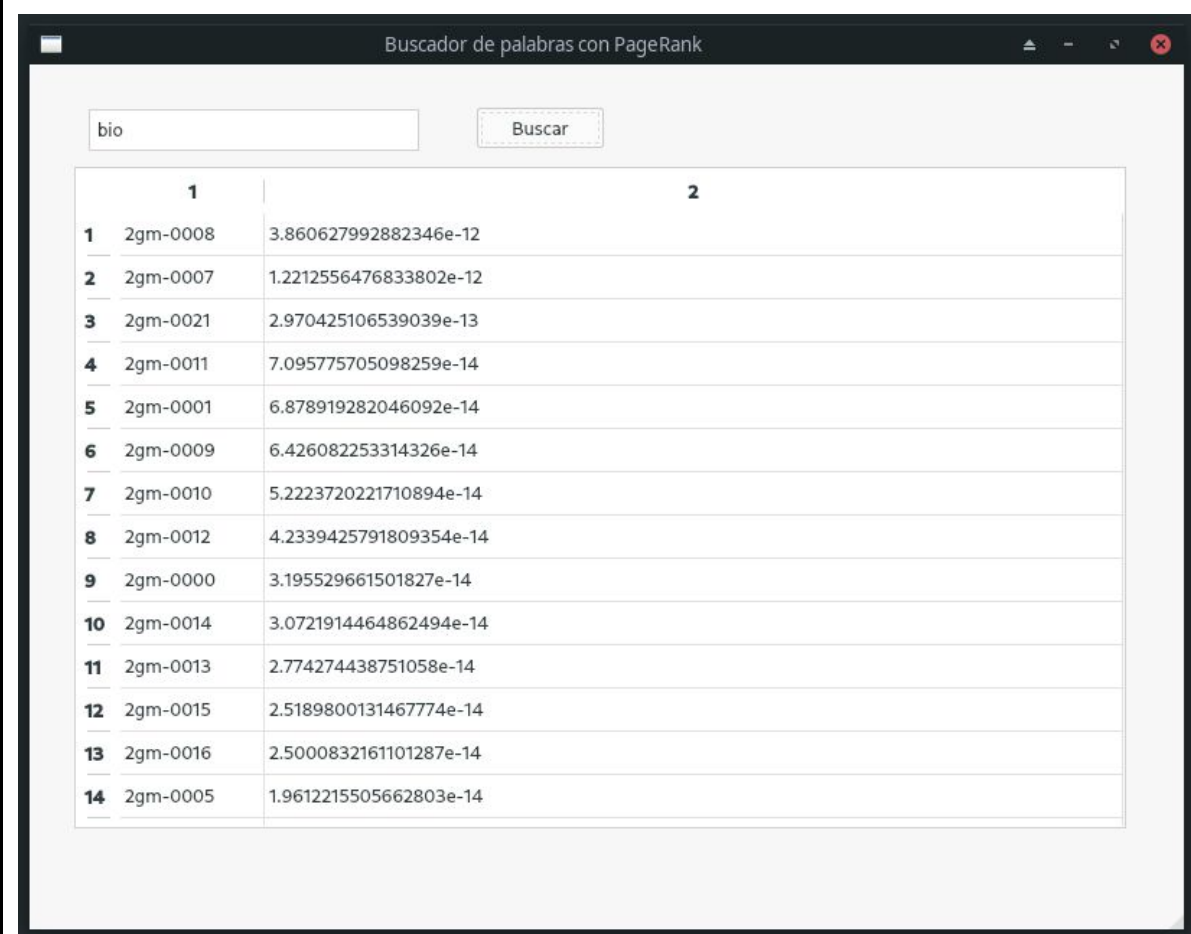
```
hadoop jar /pat/a//hadoop/tools/lib/hadoop-streaming-3.2.1.jar \
-file map.py -mapper map.py \
-file reduce.py -reducer reduce.py \
-input file:///path/to/input.txt -output output_dir
#donde este comando es utilizado para el cluster de 3 VM en ubuntu
```

## Consideraciones para la ejecución

- Cada palabra en el diccionario demora aproximadamente 20-25 minutos en obtener su índice invertido, por lo que la ejecución de 4M+ de palabras demora mucho más que el tiempo asignado; sin embargo se logró terminar palabras para demostrar el índice invertido + PageRank.
- En el Cluster de 3 nodos tras una evaluación después de varias horas tras levantar los 3 nodos se observó que estos se desconectan, tras una investigación se identificó que la causa es la tarjeta de red, por lo que no se pudo realizar los experimentos al 100%.
- Se optó por hacer una interfaz de escritorio para mostrar los resultados del trabajo.
- El archivo de comandos utilizados para cada etapa mapreduce está en la carpeta map-reduce\_scripts.

## Ejecución

Las demos de la ejecución están disponibles en el GitHub [1] para toda la implementación del cluster se siguió una guía [2] y para la ejecución de los scripts en python se siguió un tutorial base [3].



	1	2
1	2gm-0008	3.860627992882346e-12
2	2gm-0007	1.2212556476833802e-12
3	2gm-0021	2.970425106539039e-13
4	2gm-0011	7.095775705098259e-14
5	2gm-0001	6.878919282046092e-14
6	2gm-0009	6.426082253314326e-14
7	2gm-0010	5.2223720221710894e-14
8	2gm-0012	4.2339425791809354e-14
9	2gm-0000	3.195529661501827e-14
10	2gm-0014	3.0721914464862494e-14
11	2gm-0013	2.774274438751058e-14
12	2gm-0015	2.5189800131467774e-14
13	2gm-0016	2.5000832161101287e-14
14	2gm-0005	1.9612215505662803e-14

Ejecución del PageRank + Índice Invertido buscando la palabra “bio” dentro del corpus.

Buscador de palabras con PageRank		
<input type="text" value="technology"/>		<input type="button" value="Buscar"/>
1	2	
1	2gm-0018	3.677505973004457e-11
2	2gm-0007	2.6684992693524144e-12
3	2gm-0010	1.7075540472870964e-12
4	2gm-0009	1.6928522881042796e-12
5	2gm-0008	1.4754097657708334e-12
6	2gm-0029	1.2756206615246874e-12
7	2gm-0012	1.2508719447420525e-12
8	2gm-0011	1.0617424581699842e-12
9	2gm-0014	8.532643344466722e-13
10	2gm-0017	7.179067144251805e-13
11	2gm-0015	7.051818294971523e-13
12	2gm-0016	6.961309769185417e-13
13	2gm-0013	6.239199574281667e-13
14	2gm-0019	4.696250844445216e-13

Busqueda de la palabra "technology" en el corpus

## Conclusiones

En este trabajo se implementó PageRank + Índice Invertido sobre la base de datos del Corpus de Google, se desarrollaron scripts para cada etapa de mapreduce derivadas de una implementación lineal previamente realizada. Se observó que al compilar el índice invertido por palabra el tiempo de ejecución de cada consulta es de tiempo menor a 100ms. En trabajos futuros se buscara optimizar los scripts de índice invertido logrando mayor rapidez.

## Referencias

- [1] TheReverseWasp, "TheReverseWasp/PageRank\_and\_Inverted\_Index," *GitHub*. [Online]. Available: [https://github.com/TheReverseWasp/PageRank\\_and\\_Inverted\\_Index](https://github.com/TheReverseWasp/PageRank_and_Inverted_Index). [Accessed: 13-Oct-2020].
- [2] J. Torres, "How To Set Up a Hadoop 3.2.1 Multi-Node Cluster on Ubuntu 18.04 (2 Nodes)," *Medium*, 03-Feb-2020. [Online]. Available: [https://medium.com/@jootorres\\_11979/how-to-set-up-a-hadoop-3-2-1-multi-node-cluster-on-ubuntu-18-04-2-nodes-567ca44a3b12](https://medium.com/@jootorres_11979/how-to-set-up-a-hadoop-3-2-1-multi-node-cluster-on-ubuntu-18-04-2-nodes-567ca44a3b12). [Accessed: 13-Oct-2020].



[3] "Writing An Hadoop MapReduce Program In Python," *Michael G. Noll*. [Online]. Available: <https://www.michael-noll.com/tutorials/writing-an-hadoop-mapreduce-program-in-python/>. [Accessed: 13-Oct-2020].