

LABORATORIO 1: Map Reduce

Kelvin Paul Pucho Zevallos

June 21, 2023

1 Introduction

MapReduce es un enfoque de programación diseñado para facilitar la ejecución simultánea de tareas de procesamiento de datos en múltiples computadoras, especialmente cuando se trabaja con grandes conjuntos de datos y recursos informáticos básicos. El término "MapReduce" deriva de dos conceptos fundamentales en la programación funcional: "Map" y "Reduce", los cuales son métodos, macros o funciones de gran relevancia. En este trabajo se enfoca en crear las funciones Map y Reduce de manera local y comparar el rendimiento en tiempo de ejecución utilizando threads.

2 Dataset

Se empleo la dataset (1.7 Gigabytes) de Gutenberg-Text



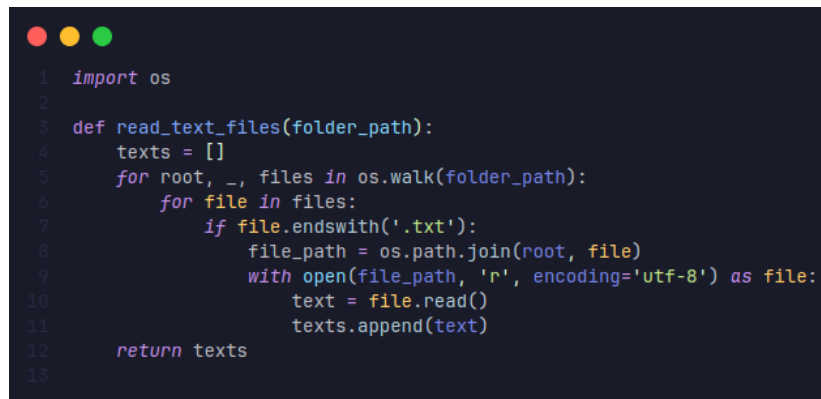
Figure 1: dataset.

3 Código

3.1 Lectura de datos

La lectura de datos se realizo de manera directa pasando por todos los archivos de una solo la vez y guardando el contenido de cada texto en una lista de textos. En la Figure 1.

3.2 Funciones MapReduce



```

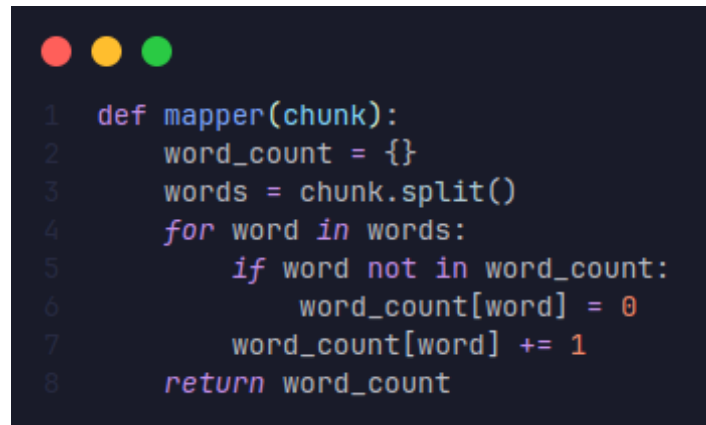
1  import os
2
3  def read_text_files(folder_path):
4      texts = []
5      for root, _, files in os.walk(folder_path):
6          for file in files:
7              if file.endswith('.txt'):
8                  file_path = os.path.join(root, file)
9                  with open(file_path, 'r', encoding='utf-8') as file:
10                     text = file.read()
11                     texts.append(text)
12      return texts
13

```

Figure 2: Read.

3.2.1 Función Map

El objetivo de la función Map es mapear los textos en su formato clave y valor por cada texto que invoco esta función.



```

1  def mapper(chunk):
2      word_count = {}
3      words = chunk.split()
4      for word in words:
5          if word not in word_count:
6              word_count[word] = 0
7          word_count[word] += 1
8      return word_count

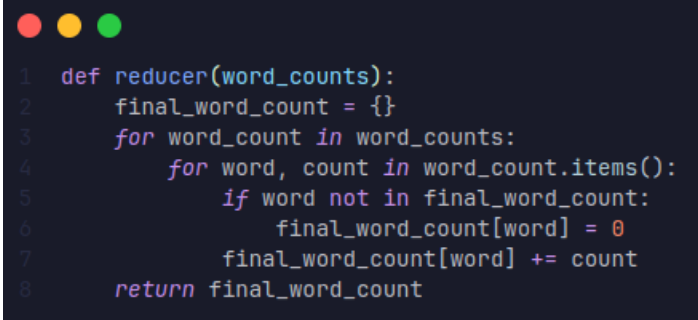
```

Figure 3: Función Map.

3.2.2 Función Reduce

En la fase de la función Reduce, los valores intermedios de esta fase se reducen para producir un único valor de salida que resume todo el conjunto de datos. Dicho valor unico consta de su clave y valor.

3.3 Funcion Worker MapReduce



```

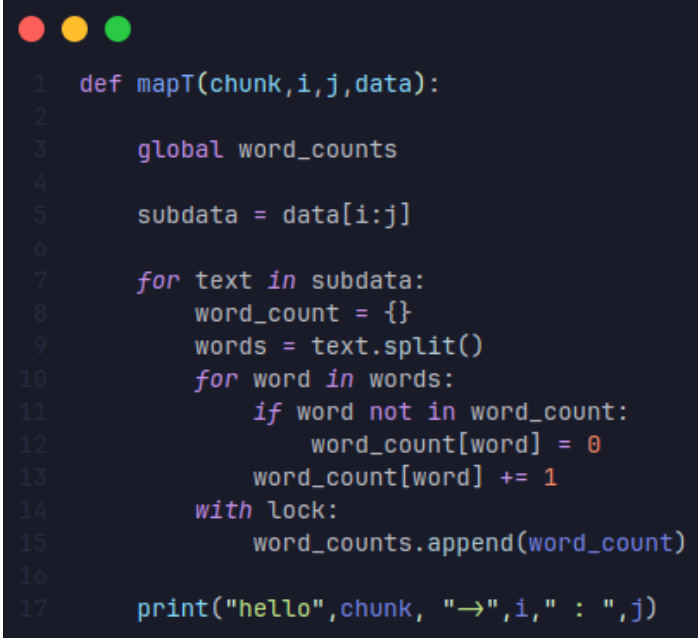
1 def reducer(word_counts):
2     final_word_count = {}
3     for word_count in word_counts:
4         for word, count in word_count.items():
5             if word not in final_word_count:
6                 final_word_count[word] = 0
7             final_word_count[word] += count
8     return final_word_count

```

Figure 4: Función Reduce.

3.3.1 Función Map

Esta fase es diferente e independiente a las otras funciones ya que esta estructurada es para una worker function para que cada threads lo ejecute en paralelo



```

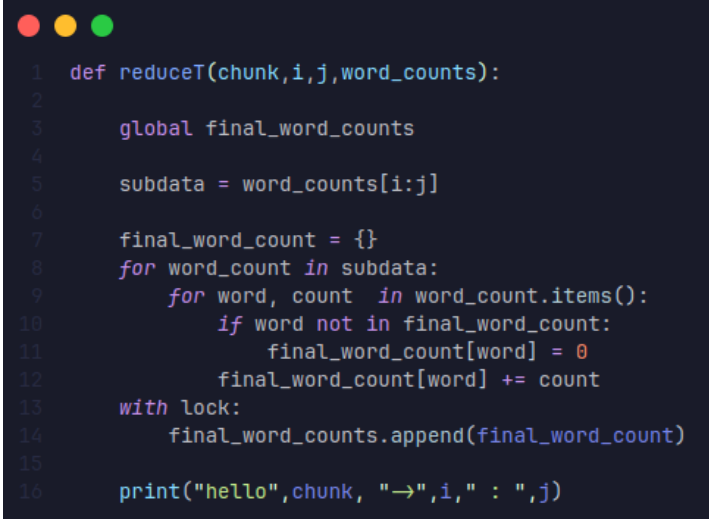
1 def mapT(chunk,i,j,data):
2
3     global word_counts
4
5     subdata = data[i:j]
6
7     for text in subdata:
8         word_count = {}
9         words = text.split()
10        for word in words:
11            if word not in word_count:
12                word_count[word] = 0
13            word_count[word] += 1
14        with lock:
15            word_counts.append(word_count)
16
17    print("hello",chunk, "→",i," : ",j)

```

Figure 5: Función Worker Map.

3.3.2 Función Reduce

Esta fase es diferente e independiente a las otras funciones ya que esta estructurada es para una worker function para que cada threads lo ejecute en paralelo



```

1  def reduceT(chunk,i,j,word_counts):
2
3      global final_word_counts
4
5      subdata = word_counts[i:j]
6
7      final_word_count = {}
8      for word_count in subdata:
9          for word, count in word_count.items():
10             if word not in final_word_count:
11                 final_word_count[word] = 0
12                 final_word_count[word] += count
13         with lock:
14             final_word_counts.append(final_word_count)
15
16     print("hello",chunk, "→",i," : ",j)

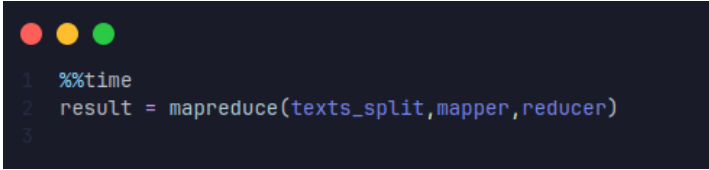
```

Figure 6: Función Worker Reduce.

4 Evaluación

4.1 Programa Secuencial

Se particiono la data en 25%, 50% y 100% y se realizo la ejecución con la función MapReduce.



```

1  %%time
2  result = mapreduce(texts_split,mapper,reducer)
3

```

Figure 7: Código de Ejecución Secuencial.

Los ejemplos que se muestran en este trabajo están estructurados de forma:

- MapReduce (Secuencial) con [25%,50% y 100%] de los datos
 - Resultados en imágenes y tiempo estimado en segundos

4.1.1 Experimentos

- MapReduce Secuencial con 25% de los datos
 - Resultados y tiempo estimado de 18 segundos

```
Output exceeds the size limit. Open
{'The': 222563,
 'Project': 64576,
 'Gutenberg': 17388,
 'eBook': 1103,
 'A': 42627,
 'Survey': 23,
 'of': 1634107,
 'Russian': 3058,
 'Literature': 41,
 'with': 493358,
 'Selections': 2,
 'by': 239956,
 'Isabel': 73,
 'Florence': 1054,
 'Hapgood': 27,
 'This': 32721,
 'eBook': 4429,
 'is': 363082,
 'for': 390172,
 'the': 3005669,
 'use': 15789
```

Figure 8: Resultados de 1000 primeros elementos

```
CPU times: user 16.8 s, sys: 322 ms, total: 17.2 s
Wall time: 18.8 s
```

Figure 9: Tiempo estimado 18 segundos

- MapReduce Secuencial con 50% de los datos
 - Resultados y tiempo estimado de 32.4 segundos

```
Output exceeds the size limit. Open
{'The': 438772,
 'Project': 129351,
 'Gutenberg': 34921,
 'eBook': 2080,
 'A': 80238,
 'Survey': 76,
 'of': 3201853,
 'Russian': 4563,
 'Literature': 86,
 'with': 887280,
 'Selections': 2,
 'by': 460432,
 'Isabel': 361,
 'Florence': 1300,
 'Hapgood': 32,
 'This': 64158,
 'eBook': 8877,
 'is': 724959,
 'for': 698837,
 'the': 5723237,
 'use': 30500
```

Figure 10: Resultados de 1000 primeros elementos

```
CPU times: user 30.9 s, sys: 300 ms, total: 31.2 s
Wall time: 32.4 s
```

Figure 11: Tiempo estimado 32.4 segundos

- MapReduce Secuencial con 100% de los datos

– Resultados y tiempo estimado de 54 segundos

```
Output exceeds the size limit. Open the full
{'The': 913708,
 'Project': 255772,
 'Gutenberg': 69433,
 'eBook': 4011,
 'A': 166521,
 'Survey': 127,
 'of': 6562332,
 'Russian': 8009,
 'Literature': 159,
 'with': 1740167,
 'Selections': 3,
 'by': 977322,
 'Isabel': 1658,
 'Florence': 2138,
 'Hapgood': 33,
 'This': 139641,
 'eBook': 17251,
 'is': 1477761,
 'for': 1441762,
 'the': 11588014}
```

Figure 12: Resultados de 1000 primeros elementos

```
CPU times: user 56 s, sys: 1.03 s, total: 57 s
Wall time: 57 s
```

Figure 13: Tiempo estimado 54 segundos

4.2 Programa Paralelo

Se particiono la data en 25%, 50% y 100% y se realizo la ejecución con la funciones processingMap y processingReduce que están orientados de forma paralela.

```
1 %%time
2 num_threads = 8
3 processingMap(texts, mapT, num_threads)
```

Figure 14: Código de Ejecución Paralelo Map.

```
1 %%time
2
3 num_threads = 8
4 processingReduce(word_counts, reduceT, num_threads)
```

Figure 15: Código de Ejecución Paralelo Reduce.

Los ejemplos que se muestran en este trabajo están estructurados de forma: Se uso el enfoque de memoria compartida para ambas funciones worker donde cada resultado de las ejecuciones representa el paralelismo de datos.

- Resultados de la Ejecución de la FuncionMap en (Paralelo) con (x) threads y el [25%,50% y 100%] de los datos
- Resultados de la Ejecución de la FuncionReduce en (Paralelo) con (x) threads y el [25%,50% y 100%] de los datos
- Resultados finales usando la funcion Reduce Secuencial
- Resultados en imágenes y tiempo estimado en segundos

4.2.1 Experimentos con 25%, 50% y 100% de los datos y con 2 threads

- Resultados de la Ejecución de la FuncionMap con 2 threads y el 25%

```
hello 1 → 423 : 846  
hello 0 → 0 : 423  
CPU times: user 14.3 s, sys: 109 ms, total: 14.4 s  
Wall time: 14.4 s
```

Figure 16: Código de Ejecución Paralelo Map en 14.4 segundos.

- Resultados de la Ejecución de la FuncionReduce con 2 threads y el 25%

```
hello 0 → 0 : 423  
hello 1 → 423 : 846  
CPU times: user 3.78 s, sys: 24.2 ms, total: 3.81 s  
Wall time: 3.8 s
```

Figure 17: Código de Ejecución Paralelo Reduce en 3.8 segundos.

- Resultados finales usando la función Reduce Secuencial

```
%%time  
result = reducer(final_word_counts)  
✓ 0.5s
```

Figure 18: Resultados en 0.5 segundos

- Resultados

```
Output exceeds the size limit. Op
{'The': 222563,
 'Project': 64576,
 'Gutenberg': 17388,
 'eBook,': 1103,
 'A': 42627,
 'Survey': 23,
 'of': 1634107,
 'Russian': 3058,
 'Literature,': 41,
 'with': 493358,
 'Selections,': 2,
 'by': 239956,
 'Isabel': 73,
 'Florence': 1054,
 'Hapgood': 27,
 'This': 32721,
 'eBook': 4429,
 'is': 363082,
 'for': 390172,
 'the': 3005669,
```

Figure 19: Resultados

- Tiempo Estimado Total: $14.4 + 3.8 + 0.5 = 18,7$ segundos
- Resultados de la Ejecución de la FuncionMap con 2 threads y el 50%

```
hello 1 → 846 : 1692
hello 0 → 0 : 846
CPU times: user 25.5 s, sys: 226 ms, total: 25.8 s
Wall time: 25.8 s
```

Figure 20: Código de Ejecución Paralelo Map en 25.8 segundos.

- Resultados de la Ejecución de la FuncionReduce con 2 threads y el 50%

```
hello 1 → 846 : 1692
hello 0 → 0 : 846
CPU times: user 7.25 s, sys: 52.1 ms, total: 7.3 s
Wall time: 7.33 s
```

Figure 21: Código de Ejecución Paralelo Reduce en 7.33 segundos.

- Resultados finales usando la función Reduce Secuencial


```
CPU times: user 774 ms, sys: 0 ns, total: 774 ms
Wall time: 778 ms
```

Figure 22: Resultados en 778 milisegundos

- Resultados

```
Output exceeds the size limit.
{'Project': 129351,
 'Gutenberg's': 517,
 'A': 80238,
 'Lowden': 8,
 'Sabbath': 404,
 'Morn,': 10,
 'by': 460432,
 'Robert': 2118,
 'Louis': 3195,
 'Stevenson': 1153,
 'This': 64158,
 'eBook': 8877,
 'is': 724959,
 'for': 698837,
 'the': 5723237,
 'use': 32502,
 'of': 3201853,
 'anyone': 10174,
 'anywhere': 5292,
```

Figure 23: Resultados

- Tiempo Estimado Total: $25.8 + 7.33 + 0.778 = 33.908$ segundos
- Resultados de la Ejecución de la FuncionMap con 2 threads y el 100%

```
hello 0 → 0 : 1691
hello 1 → 1691 : 3383
CPU times: user 42.5 s, sys: 889 ms, total: 43.4 s
Wall time: 43.3 s
```

Figure 24: Código de Ejecución Paralelo Map en 43.3 segundos.

- Resultados de la Ejecución de la FuncionReduce con 2 threads y el 100%
- Resultados finales usando la función Reduce Secuencial

```
hello 1 → 1691 : 3383
hello 0 → 0 : 1691
CPU times: user 14 s, sys: 99.3 ms, total: 14.1 s
Wall time: 14.1 s
```

Figure 25: Código de Ejecución Paralelo Reduce en 14.1 segundos.

```
%%time
result = reducer(final_word_counts)
✓ 1.6s

CPU times: user 1.45 s, sys: 170 ms, total: 1.62 s
Wall time: 1.63 s
```

Figure 26: Resultados en 1.63 segundos

- Resultados

```
Output exceeds the size limit. Open the full output data.
{'The': 913708,
 'Project': 255772,
 'Gutenberg': 69433,
 'EBook': 4234,
 'of': 6562332,
 'An': 18474,
 'Old': 18558,
 'Maid': 246,
 'by': 977322,
 'Honore': 514,
 'de': 89172,
 'Balzac': 855,
 'This': 139641,
 'eBook': 17251,
 'is': 1477761,
 'for': 1441762,
 'the': 11588014,
 'use': 67235,
 'anyone': 21728,
 'anywhere': 10132,
 'at': 1164283,
```

Figure 27: Resultados

- Tiempo Estimado Total: $43.3 + 14.1 + 1.63 = 59.03$ segundos

4.2.2 Experimentos con 25%, 50% y 100% de los datos y con 8 threads

- Resultados de la Ejecución de la FuncionMap con 8 threads y el 25%

```
hello 7 → 735 : 846
hello 6 → 630 : 735
hello 5 → 525 : 630
hello 1 → 105 : 210
hello 4 → 420 : 525
hello 2 → 210 : 315
hello 0 → 0 : 105
hello 3 → 315 : 420
CPU times: user 12.4 s, sys: 247 ms, total: 12.7 s
Wall time: 12.5 s
```

Figure 28: Código de Ejecución Paralelo Map en 12.5 segundos.

- Resultados de la Ejecución de la FuncionReduce con 8 threads y el 25%

```
hello 5 → 525 : 630
hello 0 → 0 : 105
hello 1 → 105 : 210
hello 2 → 210 : 315
hello 3 → 315 : 420
hello 7 → 735 : 846
hello 6 → 630 : 735
hello 4 → 420 : 525
CPU times: user 3.26 s, sys: 53.1 ms, total: 3.31 s
Wall time: 3.25 s
```

Figure 29: Código de Ejecución Paralelo Reduce en 3.25 segundos.

- Resultados finales usando la función Reduce Secuencial

```
%%time
result = reducer(final_word_counts)
✓ 0.9s

CPU times: user 865 ms, sys: 11.9 ms, total: 877 ms
Wall time: 876 ms
```

Figure 30: Resultados en 0.876 segundos

- Resultados

```
Output exceeds the size limit. 0
{'The': 222563,
 'Project': 64576,
 'Gutenberg': 17388,
 'eBook': 1103,
 'Records': 21,
 'of': 1634107,
 'a': 1171937,
 'Family': 270,
 'Engineers': 22,
 'by': 239956,
 'Robert': 1311,
 'Louis': 1344,
 'Stevenson': 1085,
 'This': 32721,
 'eBook': 4429,
 'is': 363082,
 'for': 390172,
 'the': 3005669,
 'use': 15709,
 'anyone': 5116,
 'anywhere': 2635,
 'at': 326445,
 'no': 136862,
 'cost': 3840,
 'and': 1741370}
```

Figure 31: Resultados

- Tiempo Estimado Total: $12.5 + 3.25 + 0.876 = 16.626$ segundos
- Resultados de la Ejecución de la FuncionMap con 8 threads y el 50%

```
hello 6 → 1266 : 1477
hello 4 → 844 : 1055
hello 2 → 422 : 633
hello 3 → 633 : 844
hello 5 → 1055 : 1266
hello 7 → 1477 : 1692
hello 1 → 211 : 422
hello 0 → 0 : 211
CPU times: user 21.6 s, sys: 479 ms, total: 22.1 s
Wall time: 21.8 s
```

Figure 32: Código de Ejecución Paralelo Map en 21.8 segundos.

- Resultados de la Ejecución de la FuncionReduce con 8 threads y el 50%

```
hello 6 → 1266 : 1477
hello 1 → 211 : 422
hello 0 → 0 : 211
hello 7 → 1477 : 1692
hello 2 → 422 : 633
hello 5 → 1055 : 1266
hello 4 → 844 : 1055
hello 3 → 633 : 844
CPU times: user 6.35 s, sys: 106 ms, total: 6.46 s
Wall time: 6.34 s
```

Figure 33: Código de Ejecución Paralelo Reduce en 6.34 segundos.

- Resultados finales usando la función Reduce Secuencial

```
%%time
result = reducer(final_word_counts)
✓ 1.4s

CPU times: user 1.35 s, sys: 35.5 ms, total: 1.38 s
Wall time: 1.38 s
```

Figure 34: Resultados en 1.38 segundos

- Resultados

```
Output exceeds the size limit. Open the full output
{'The': 438772,
 'Project': 129351,
 'Gutenberg': 34921,
 'eBook': 2080,
 'Dutch': 1190,
 'Courage': 75,
 'and': 3252272,
 'Other': 1919,
 'Stories': 172,
 'by': 460432,
 'Jack': 5085,
 'London': 7564,
 'This': 64158,
 'eBook': 8877,
 'is': 724959,
 'for': 698837,
 'the': 5723237,
 'use': 32502,
 'of': 3201853,
 'anyone': 10174,
 'anywhere': 5292,
 'at': 595442,
 'no': 249486,
 'cost': 7539,
```

Figure 35: Resultados

- Tiempo Estimado Total: $21.8 + 6.34 + 1.38 = 29.52$ segundos

- Resultados de la Ejecución de la FuncionMap con 8 threads y el 100%

```
hello 5 → 2110 : 2532
hello 3 → 1266 : 1688
hello 7 → 2954 : 3383
hello 1 → 422 : 844
hello 4 → 1688 : 2110
hello 2 → 844 : 1266
hello 6 → 2532 : 2954
hello 0 → 0 : 422
CPU times: user 43 s, sys: 1.04 s, total: 44.1 s
Wall time: 43.6 s
```

Figure 36: Código de Ejecución Paralelo Map en 43.6 segundos.

- Resultados de la Ejecución de la FuncionReduce con 8 threads y el 100%

```
hello 1 → 422 : 844
hello 0 → 0 : 422
hello 3 → 1266 : 1688
hello 2 → 844 : 1266
hello 7 → 2954 : 3383
hello 6 → 2532 : 2954
hello 5 → 2110 : 2532
hello 4 → 1688 : 2110
CPU times: user 13.6 s, sys: 210 ms, total: 13.8 s
Wall time: 13.6 s
```

Figure 37: Código de Ejecución Paralelo Reduce en 13.6 segundos.

- Resultados finales usando la función Reduce Secuencial

```
%%time
result = reducer(final_word_counts)
✓ 2.4s
CPU times: user 2.38 s, sys: 79.7 ms, total: 2.46 s
Wall time: 2.46 s
```

Figure 38: Resultados en 2.46 segundos

- Resultados

```

Output exceeds the size limit. Up
{'The': 913708,
 'Project': 255772,
 'Gutenberg': 69433,
 'EBook': 4234,
 'of': 6562332,
 'Stories': 996,
 'Animal': 147,
 'Sagacity': 4,
 'by': 977322,
 'W.H.G.': 23,
 'Kingston': 141,
 'This': 139641,
 'eBook': 17251,
 'is': 1477761,
 'for': 1441762,
 'the': 11588014,
 'use': 67235,
 'anyone': 21728,
 'anywhere': 10132,
 'at': 1164283,
 'no': 492807,
 'cost': 16201,
 'and': 6250891,
 'with': 1740167,
 'almost': 81049,

```

Figure 39: Resultados

- Tiempo Estimado Total: $43.6 + 13.6 + 2.46 = 59.66$ segundos

5 Conclusion

Los codigos en forma secuencial consumieron mas tiempo en 25% y 50%, ya que la diferencia era de 4 a 5 segundos menos con un código en paralelo usando 8 threads, pero con 2 threads los tiempos eran los mismos e incluso mayores, por otro lado respecto a un código secuencial usando 100% de datos fue mas rápido con una diferencia de 4 a 5 segundos mas que un código en paralelo con 8 threads esto debido a que el código en este trabajo paraleliza dos veces uno en la operación del Map y otro en la operación del Reduce pero también usa una parte secuencial para unir los resultados de la operación Reduce que depende de la cantidad de threads, por lo tanto la creación y la unión mas los bloques en la sección critica hace que el código sea mas lento que el código secuencial.