

# **Proyecto Final**

## **Procesamiento de Big Graph con MapReduce y Giraph**

Arellano E. Nahuel  
nahuel.arellano@gmail.com

### **RESUMEN**

Este informe presenta una comparativa de rendimiento y de escalabilidad entre las herramientas Hadoop MapReduce y Apache Giraph para el procesamiento de grandes volúmenes de información o Big Data, en particular para el procesamiento de Big Graph.

En este trabajo se presenta un análisis de costo en tiempo de ejecución del PageRank, algoritmo que permite clasificar páginas web según su relevancia, los experimentos muestran que Giraph tiene un desempeño muy superior sobre el de MapReduce.

### **INTRODUCCIÓN**

En la actualidad, las organizaciones de todo tipo y tamaño tienen a su disposición grandes volúmenes de información a muy bajo costo.

IBM ha revelado que cada día se generan más de 2,5 quintillones de bytes de información [1]. De forma similar, International Data Corporation IDC, estima que el tamaño del universo digital era de 4.4 ZB en el 2013 y pronostica un crecimiento de diez veces de este valor para el 2020 [2]. Ante este crecimiento exponencial de la información, se popularizó el término Big Data. Según Gartner [3], Big Data se define como: grandes volúmenes, alta velocidad y gran variedad de información que demandan formas rentables e innovadoras de procesamiento de la información que permiten un mejor conocimiento, toma de decisiones y automatización de procesos.

Aun cuando exista una tendencia al aumento en los volúmenes de espacio para el almacenamiento de información, no hay un aumento equivalente en la velocidad de acceso y procesamiento de grandes volúmenes de información. Una solución directa a este problema, para reducir el tiempo de acceso y procesamiento de datos, es el uso de clúster de computadoras. En este contexto, existen varios sistemas distribuidos que permiten combinar datos desde múltiples fuentes, pero con un alto costo de análisis y desarrollo debido principalmente a la concurrencia, sincronización de tareas, acceso y transferencia de datos.

Aquí es donde Hadoop [4], un framework opensource basado en MapReduce [5] es de gran importancia y utilidad.

El procesamiento de Big Data está muy asociado a información no estructurada, esto es, datos directamente no relacionados. Sin embargo,

en las redes sociales actuales, como Facebook y Twitter, existen vínculos entre los nodos componentes de las mismas, para lo cual su representación directa como grafos es adecuada. Sin embargo, por la estructura y asociaciones de grafos, MapReduce y Hadoop no son óptimos para su procesamiento, por el alto costo de la entrada/salida de información y la posibilidad de requerir muchas fases de MapReduce encadenadas. En este contexto, Giraph es una contraparte opensource de Pregel, sistema de procesamiento de grafos desarrollado por Google, para el procesamiento de grandes grafos.

El principal objetivo de este trabajo, es presentar una revisión de la aplicación de Hadoop y Giraph para el procesamiento de grafos, y así resaltar las ventajas prácticas que tiene Giraph sobre Hadoop al momento de resolver problemas con datos relacionados y que usualmente se solucionan mediante algoritmos iterativos, como son las estructuras de grafos. Para esto se realizará una comparativa entre Hadoop y Giraph del rendimiento de soluciones algorítmicas a un problema clasificación de páginas web.

## **OBJETIVOS**

El presente trabajo persigue una diversidad de objetivos. Los principales están vinculados a la investigación académica. También hay un conjunto de objetivos secundarios que surgen a raíz de no contar con experiencia en Hadoop ni en tecnologías de Big Data, ni con la infraestructura necesaria para las pruebas.

### **Objetivos principales**

- Implementar el algoritmo PageRank en un entorno distribuido, sobre las tecnologías Hadoop MapReduce y Apache Giraph.
- Medir la eficiencia del algoritmo PageRank sobre MapReduce y Apache Giraph para procesar gran volumen de información con diferentes configuraciones de cluster.

### **Objetivos secundarios**

- Construcción de un grupo de clusters para desarrollar las pruebas utilizando un proveedor en la nube.
- Desarrollar e implementar en MapReduce el algoritmo PageRank.
- Implementar y configurar Apache Giraph para la ejecución del algoritmo PageRank.

## ESTADO DEL ARTE

En esta sección se fundamenta el marco teórico del trabajo, se desarrolla las disciplinas mencionadas y se introducen los conceptos tenidos en cuenta en el diseño del algoritmo.

Se comienza detallando la plataforma de software sobre la cual se implementan estos procesos, Hadoop<sup>1</sup>. Luego se detalla las principales ideas y conceptos abordados en el trabajo, MapReduce y Apache Giraph<sup>2</sup>.

### Hadoop

El proyecto Apache Hadoop desarrolla software de código abierto para una informática confiable, escalable y distribuida [4].

La biblioteca de software Apache Hadoop es un marco que permite el procesamiento distribuido de grandes conjuntos de datos en clústeres de computadoras que usan modelos de programación simples.

Está diseñado para escalar desde servidores únicos a miles de máquinas, cada una de las cuales ofrece cómputo y almacenamiento local. En lugar de confiar en el hardware para ofrecer alta disponibilidad, la biblioteca está diseñada para detectar y manejar fallas en la capa de aplicaciones, por lo que entrega un servicio altamente disponible sobre un grupo de computadoras, cada una de las cuales puede ser propensa a fallas.

El proyecto incluye estos módulos:

- Hadoop Common: las utilidades comunes que son compatibles con los otros módulos de Hadoop.
- Sistema de archivos distribuidos de Hadoop (HDFS): un sistema de archivos distribuidos que proporciona acceso de alto rendimiento a los datos de la aplicación.
- HADOOP YARN: un marco para la programación de trabajos y la administración de recursos del clúster.
- Hadoop MapReduce: un sistema basado en YARN para el procesamiento paralelo de grandes conjuntos de datos.

### MapReduce

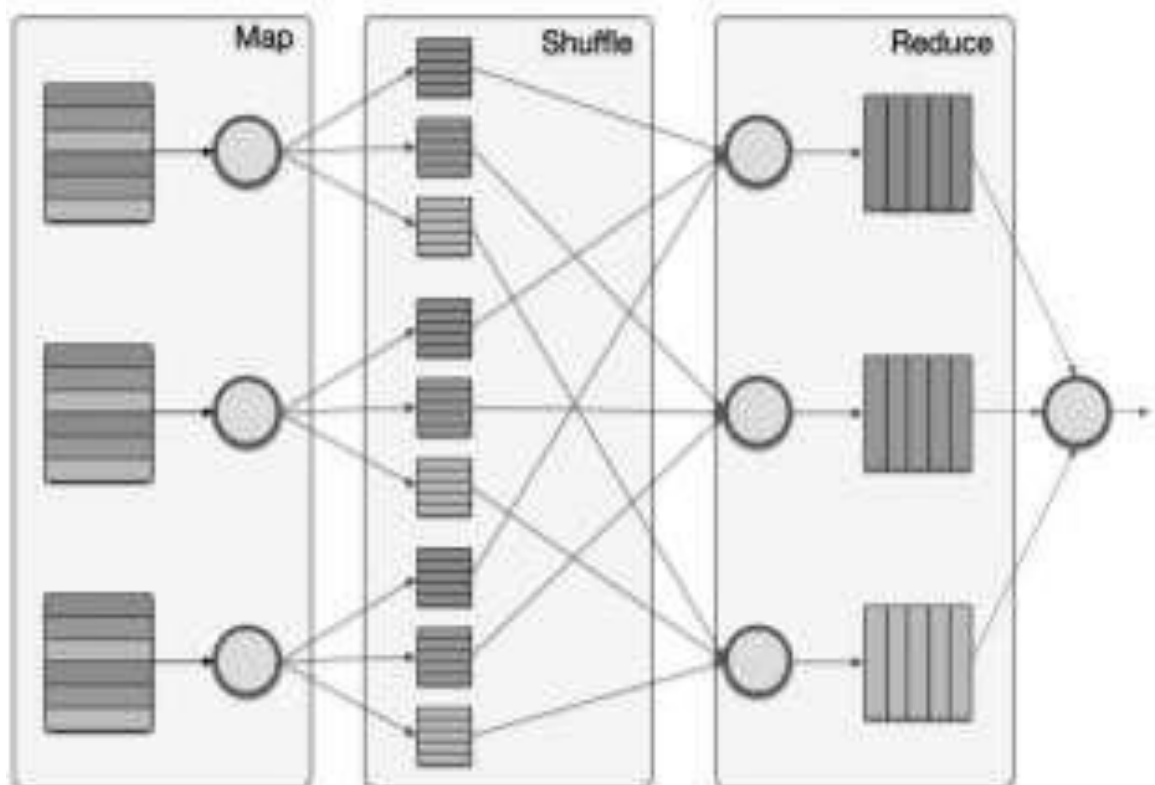
Hadoop implementa un paradigma computacional llamado MapReduce [5] Algorítmicamente está basado en el concepto de “divide y vencerás”, es decir, divide el problema en pequeños trozos para su procesamiento en paralelo y así obtener soluciones en un entorno distribuido. MapReduce toma un conjunto de pares key-value iniciales, y produce un conjunto final de pares key-value, y su modelo de programación está compuesto por una función Map y una función Reduce: Map recibe un par key-value de entrada y produce un conjunto de pares key-value intermedios, entonces

<sup>1</sup> <http://hadoop.apache.org>

<sup>2</sup> <http://giraph.apache.org>

MapReduce agrupa todos los valores intermedios asociados con la misma key intermedia como entrada de la función Reduce. De esta forma, la función Reduce acepta una key intermedia y un conjunto de valores para esa key con el fin de mezclar estos valores y así formar un nuevo conjunto de salida final.

Normalmente, una solución en MapReduce se realiza en cinco etapas. Concretamente, i) splitting, los datos se dividen en múltiples partes y se entregan a cada mapper; ii) mapper, ejecuta la función map encargada de procesar los datos; iii) combiner, funciona directamente sobre la salida de los mappers para la agregación local; iv) shuffle, responsable de barajar y ordenar los pares key-value para la función reduce, y v) reduce, en el último paso se reducen todos los valores con la misma key intermedia, para generar pares key-value finales.



*Ilustración 1*

## Giraph

Giraph es un framework para el procesamiento de grafos de gran tamaño sobre una plataforma distribuida, con una alta tolerancia a fallas. En términos prácticos, es una implementación libre de Google Pregel, la arquitectura de procesamiento de grafos desarrollada por Google [6]. Giraph se ejecuta como un trabajo de sólo mapping sobre el entorno distribuido Hadoop.

Desde su lanzamiento en 2013, Giraph ha sido empleado en diversos dominios de aplicación y ha sido considerado en diversas evaluaciones empíricas de los sistemas de procesamiento para grafos. Una buena prueba de la capacidad de Giraph para procesar grafos de gran tamaño es su uso en Facebook.

Giraph hereda de Pregel un modelo de programación centrado en los vértices (vertex-centric programming model), en el cual cada vértice puede intercambiar mensajes con otros vértices en una secuencia de iteraciones. En ningún momento el usuario debe lidiar con la complejidad de programación de un sistema de procesamiento paralelo y/o distribuido, incluida la distribución de los datos sobre el cluster o la forma en que se implementa la tolerancia a fallas. De esta manera, el programador se concentra en los aspectos específicos del algoritmo, mientras que Giraph se encarga de ejecutar el algoritmo en paralelo y en una plataforma distribuida de manera transparente, obteniendo aplicaciones escalables, descentralizadas y masivamente paralelas.

Si bien el paradigma de programación de Giraph permite una implementación más sencilla de algoritmos sobre grafos, esto en comparación con los sistemas de programación paralela/distribuida tradicionales. Su uso no es tan sencillo como se argumenta. Sólo la instalación y configuración inicial de Giraph, considerando las bibliotecas involucradas y la plataforma distribuida, toma como mínimo 30 minutos, y requiere de conocimientos intermedios de administración de sistemas y de maven. La ejecución de un algoritmo se puede resumir en tres pasos: carga de datos, ejecución del algoritmo, y procesamiento del resultado.

La carga de datos implica transformar los datos originales a alguno de los formatos de texto soportados por Giraph, tarea que puede tomar bastante tiempo dependiendo de la estructura de los datos y el formato elegido. Como se indicó anteriormente, los algoritmos deben ser programados centrándose en los vértices. Esto puede parecer sencillo, sin embargo, es un gran desafío adaptarse a este nuevo paradigma de programación ya que es muy distinto a la forma tradicional de diseñar algoritmos para grafos. Si bien las funciones básicas del API de Giraph son fáciles de entender, existen algunas funciones que no son intuitivas y para las cuales existe poca documentación. Finalmente, el resultado de ejecutar Giraph se traduce en un archivo plano que contiene una línea de texto para cada nodo/etiqueta en el grafo. Claramente, este tipo de resultado implica un procesamiento y análisis adicional.

## PageRank

PageRank es una marca registrada y patentada por Google el 9 de enero de 1999 que ampara una familia de algoritmos utilizados para asignar de forma numérica la relevancia de los documentos (o páginas web) indexados [7].

El sistema PageRank es utilizado por el popular motor de búsqueda Google para ayudarle a determinar la importancia o relevancia de una página. Fue desarrollado por los fundadores de Google, Larry Page y Sergey Brin, en la Universidad de Stanford mientras estudiaban el posgrado en ciencias de la computación.

PageRank confía en la naturaleza democrática de la web utilizando su vasta estructura de enlaces como un indicador del valor de una página en concreto.

PageRank interpreta un enlace de una página A a una página B como un voto, de la página A, para la página B. Pero PageRank mira más allá del volumen de votos, o enlaces que una página recibe; también analiza la página que emite el voto.

Los votos emitidos por las páginas consideradas "importantes", es decir con un PageRank elevado, valen más, y ayudan a hacer a otras páginas "importantes".

Formula:

$$PR(A) = (1 - d) + d \sum_{i=1}^n \frac{PR(i)}{C(i)}$$

Donde:

- $PR(A)$  es el PageRank de la página A.
- $d$  es un factor de amortiguación que tiene un valor entre 0 y 1.
- $PR(i)$  son los valores de PageRank que tienen cada una de las páginas  $i$  que enlazan a A.
- $C(i)$  es el número total de enlaces salientes de la página  $i$  (sean o no hacia A).

### *Ilustración 2*

Algunos expertos aseguran que el valor de la variable  $d$  suele ser 0,85. Representa la probabilidad de que un navegante continúe pulsando links al navegar por Internet en vez de escribir una url directamente en la barra de direcciones o pulsar uno de sus marcadores y es un valor establecido por Google. Por lo tanto, la probabilidad de que el usuario deje de pulsar links y navegue directamente a otra web aleatoria es  $1-d$  [8]. La introducción del factor de amortiguación en la fórmula resta algo de peso a todas las páginas

de Internet y consigue que las páginas que no tienen enlaces a ninguna otra página no salgan especialmente beneficiadas. Si un usuario aterriza en una página sin enlaces, lo que hará será navegar a cualquier otra página aleatoriamente, lo que equivale a suponer que una página sin enlaces salientes tiene enlaces a todas las páginas de Internet.

La calidad de la página y el número de posiciones que ascienda se determina por una "votación" entre todas las demás páginas de la World Wide Web acerca del nivel de importancia que tiene esa página. Un hiperenlace a una página cuenta como un voto de apoyo. El PageRank de una página se define recursivamente y depende del número y PageRank de todas las páginas que la enlazan. Una página que está enlazada por muchas páginas con un PageRank alto consigue también un PageRank alto. Si no hay enlaces a una página web, no hay apoyo a esa página específica. El PageRank de la barra de Google va de 0 a 10. Diez es el máximo PageRank posible y son muy pocos los sitios que gozan de esta calificación, 1 es la calificación mínima que recibe un sitio normal, y cero significa que el sitio ha sido penalizado o aún no ha recibido una calificación de PageRank.

## EXPERIMENTOS Y RESULTADOS

### Dataset

Para realizar las pruebas de rendimiento se utilizaron los grafos dirigidos de Wikipedia<sup>3</sup> y LiveJournal online social network<sup>4</sup>.

Según los experimentos realizados, los grafos deben tener un volumen superior a 1 GB para que califiquen como dataset válidos.

	Alias	Nodes	Edges	Volumen
<b>Wikipedia</b>	<b>D1</b>	2.790.239	104.673.033	1,5 GB
<b>Social Live journal</b>	<b>D2</b>	4.847.571	68.993.773	1,1 GB

*Tabla 1*

	D1	D2
<b>L<sup>5</sup></b>	104.673.033	68.993.773
<b>g<sup>6</sup></b>	2.790.239	4.847.571
<b>g(g-1)<sup>7</sup></b>	7,78543E+12	2,34989E+13
<b>Δ<sup>8</sup></b>	1,34447E-05	2,93604E-06

*Tabla 2*

### Descripción de los dataset

#### Wikipedia

Este conjunto de datos contiene rutas de navegación humanas en Wikipedia. En Wikispeedia, se solicita a los usuarios que naveguen desde una fuente determinada a un artículo de destino dado, haciendo clic solo en los enlaces de Wikipedia [9]

<sup>3</sup> <http://networkrepository.com/web-wikipedia-link-it.php>

<sup>4</sup> <https://snap.stanford.edu/data/soc-LiveJournal1.html>

<sup>5</sup> L: Cantidad de aristas

<sup>6</sup> g: Cantidad de nodos

<sup>7</sup> g(g-1): Número de arcos posibles

<sup>8</sup> Δ: Densidad de grafos dirigidos  $\Delta = L / g(g-1)$  [11]



## Social Live journal

LiveJournal es una comunidad en línea gratuita con casi 10 millones de miembros; una fracción significativa de estos miembros son altamente activos. (Por ejemplo, aproximadamente 300,000 actualizan su contenido en un período de 24 horas). LiveJournal permite a los miembros mantener diarios, blogs individuales y grupales, y permite a las personas declarar qué otros miembros son sus amigos a los que pertenecen [10].

## Formato de la colección

El dataset originalmente es un archivo plano compuesto de la siguiente forma:

```
source_id dest_id
```

```
1 2
1 3
1 4
2 1
2 4
3 3
4 2
4 3
```

Para los experimentos se tuvo que transformar los dataset al formato propuesto por Giraph<sup>9</sup>.

```
[source_id, source_value, [[dest_id, edge_value],...]]

[1,0.25, [[2,1], [3,1], [4,1]]]
[2,0.25, [[1,1], [4,1]]]
[3,0.25, [[3,1]]]
[4,0.25, [[2,1], [3,1]]]
```

- Se desarrollaron los scripts **src/sortMapper.py** y **src/sortReducer.py** para realizar dicha transformación.

## Construcción del Clúster

Al momento de realizar los experimentos se optó por construir los cluster en GCP<sup>10</sup> (Google Cloud Platform), este proveedor cuenta con un periodo

<sup>9</sup> <http://giraph.apache.org/io.html>

<sup>10</sup> <https://console.cloud.google.com>

de prueba gratuito y brinda los recursos suficientes para realizar las pruebas pertinentes.

Para poder realizar la instalación de los clusters, primero se tuvo que registrar una cuenta en GCP e instalar el paquete de herramientas gcloud-sdk.

```
nahuel@nahuel-VirtualBox ~ $ sudo apt-get install google-cloud-sdk
```

#### Vincular cuenta y proyecto:

```
nahuel@nahuel-VirtualBox ~ $ gcloud init
Welcome! This command will take you through the configuration of gcloud.
```

```
Settings from your current configuration [default] are:
core:
  account: nahuel.arellano@gmail.com
  disable_usage_reporting: 'False'
  project: taller-ii-202623
```

```
Pick configuration to use:
[1] Re-initialize this configuration [default] with new settings
[2] Create a new configuration
Please enter your numeric choice:
```

Una vez configurada la cuenta se procede a la construcción de cuatro clusters, por medio de las siguientes líneas de comandos:

```
#Single node
gcloud dataproc --region asia-east1 clusters create
cluster-arellanon-0 --subnet default --zone asia-east1-a --single-node --master-machine-type custom-1-51200-ext --master-boot-disk-size 500 --project taller-ii-202623
```

#### Master

- CPU: 1 CPU, 1 Core - Intel(R) Xeon(R) CPU @ 2.50GHz
- RAM: 50GB
- DISCO: HDD 500GB

```
#1 Master - 2 Workers
gcloud dataproc --region southamerica-east1 clusters
create cluster-arellanon-2 --subnet default --zone
southamerica-east1-a --master-machine-type n1-highmem-2
--master-boot-disk-size 100 --num-workers 2 --worker-
machine-type custom-1-32768-ext --worker-boot-disk-size
100 --project taller-ii-202623
```

#### Master

- CPU: 2 CPU, 1 Core - Intel(R) Xeon(R) CPU @ 2.20GHz
- RAM: 13GB
- DISCO: HDD 100GB

#### Workers

- CPU: 1 CPU, 1 Core - Intel(R) Xeon(R) CPU @ 2.20GHz
- RAM: 32GB
- DISCO: HDD 100GB

```
#1 Master - 4 Workers
gcloud dataproc --region us-central1 clusters create
cluster-arellanon-4 --subnet default --zone us-cen-
trall1-a --master-machine-type n1-highmem-2 --master-
boot-disk-size 100 --num-workers 4 --worker-machine-
type custom-1-16384-ext --worker-boot-disk-size 100 --
project taller-ii-202623
```

#### Master

- CPU: 2 CPU, 1 Core - Intel(R) Xeon(R) CPU @ 2.60GHz
- RAM: 13GB
- DISCO: HDD 100GB

#### Workers

- CPU: 1 CPU, 1 Core - Intel(R) Xeon(R) CPU @ 2.60GHz
- RAM: 16GB
- DISCO: HDD 100GB

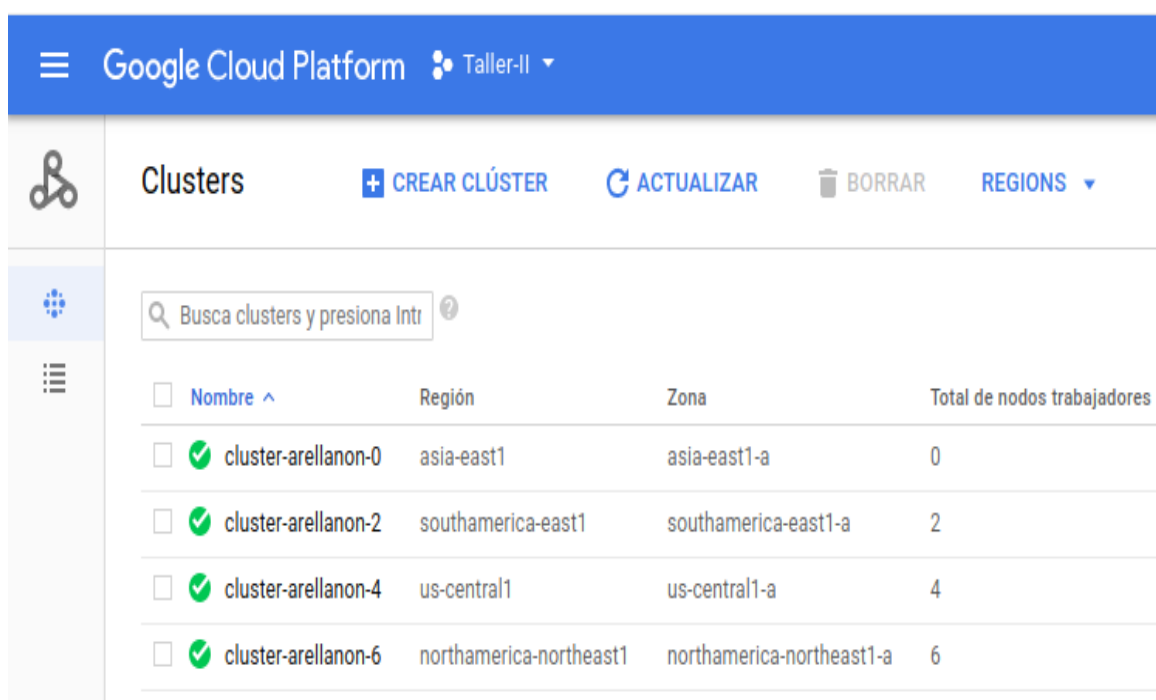
```
#1 Master - 6 Workers
gcloud dataproc --region northamerica-northeast1 clus-
ters create cluster-arellanon-6 --subnet default --zone
northamerica-northeast1-a --master-machine-type n1-
highmem-2 --master-boot-disk-size 100 --num-workers 6 -
--worker-machine-type custom-1-16384-ext --worker-boot-
disk-size 100 --project taller-ii-202623
```

### Master

- CPU: 2 CPU, 1 Core - Intel(R) Xeon(R) CPU @ 2.60GHz
- RAM: 13GB
- DISCO: HDD 100GB

### Workers

- CPU: 1 CPU, 1 Core - Intel(R) Xeon(R) CPU @ 2.60GHz
- RAM: 16GB
- DISCO: HDD 100GB



The screenshot shows the Google Cloud Platform interface for managing clusters. The header bar is blue with the Google Cloud Platform logo and the user 'Taller-II'. Below the header, the 'Clusters' section is active, showing a search bar and buttons for '+ CREAR CLÚSTER', 'ACTUALIZAR', 'BORRAR', and 'REGIONS'. A table lists five clusters, each with a checkbox, a green checkmark, a name, a region, a zone, and a total number of worker nodes.

<input type="checkbox"/>	Nombre ^	Región	Zona	Total de nodos trabajadores
<input type="checkbox"/>	✓ cluster-arellanon-0	asia-east1	asia-east1-a	0
<input type="checkbox"/>	✓ cluster-arellanon-2	southamerica-east1	southamerica-east1-a	2
<input type="checkbox"/>	✓ cluster-arellanon-4	us-central1	us-central1-a	4
<input type="checkbox"/>	✓ cluster-arellanon-6	northamerica-northeast1	northamerica-northeast1-a	6

*Ilustración 3*

El Sistema Operativo de los equipos es Debian GNU/Linux 8 y sobre los mismos viene instalada la versión 2.8.3 de Hadoop.

### **Desarrollo**

En esta sección se describe el proyecto PageRank-Hadoop que fue desarrollado para la ejecución de los experimentos.

El proyecto contiene varios scripts y fue diseñado para ejecutar las pruebas de forma automática, con la intención de ahorrar tiempos de ejecución y costos en los clusters.

PageRank-Hadoop contiene los siguientes archivos:

```
.
├── config.sh
├── download_files.sh
├── giraph-run-time.sh
├── hadoop-run-converge.sh
├── hadoop-run-time.sh
├── Instalacion de Cluster.txt
├── install_giraph.sh
├── install.sh
├── project_run.sh
├── README.md
├── src
│   ├── config.py
│   ├── config.pyc
│   ├── MaxDiff.py
│   ├── prMapper.py
│   ├── prReducer.py
│   ├── sortMapper.py
│   ├── sort.py
│   └── sortReducer.py
├── test_giraph-run.sh
├── test.sh
├── test-times.sh
└── tmp
    ├── nodes
    └── values
```

A continuación, se describirán los scripts más relevantes:

**install.sh:** Es el archivo principal, necesita como parámetro la cantidad de workers. Su función es instalar los programas necesarios para el proyecto, descargar los archivos de prueba y ejecutar el proyecto.

```
#!/usr/bin/env bash
if [ -z "$1" ]; then
echo "Ingrese nro de workers"
exit
fi
workers=$1
sudo apt-get install bc -y
./install_giraph.sh
./download_files.sh
./project_run.sh $workers
```

**install\_giraph.sh:** Este script es el encargado de descargar, generar variables de entornos, crear enlaces simbólicos e instalar los programas necesarios para poder utilizar giraph.

Giraph se descarga desde mi repositorio personal, no desde el repositorio oficial, porque se tuvo que modificar la cantidad de iteraciones máximas para el programa SimplePageRankComputation, sino por defecto está limitado a 30 iteraciones.

```
#!/usr/bin/env bash
sudo apt-get install maven -y
cd /usr/local/
#sudo git clone https://github.com/apache/giraph.git
sudo git clone https://github.com/arellanon/giraph
sudo chown -R nahuel:nahuel giraph/
echo "export HADOOP_HOME=/usr/lib/hadoop" >> ~/.bashrc
echo "export GIRAPH_HOME=/usr/local/giraph" >>
~/.bashrc
source ~/.bashrc
HADOOP_HOME=/usr/lib/hadoop
GIRAPH_HOME=/usr/local/giraph
cd $GIRAPH_HOME
#Se elimina bug
rm giraph-block-app-8/src/test/java/org/apache/giraph/block_app/framework/no_vtx/MessagesWithoutVerticesTest.java
#Se ejecuta la instalacion de GIRAPH
mvn -Phadoop_2 -Dhadoop.version=2.8.2 -DskipTests clean
package

#Creamos enlace simbolico
ln -s /usr/local/giraph/giraph-examples/target/giraph-examples-1.3.0-SNAPSHOT-for-hadoop-2.8.2-jar-with-dependencies.jar /usr/local/giraph/giraph-examples/target/giraph-examples-with-dependencies.jar
```

**project\_run.sh:** Recibe como parámetro el número de workers. Se encarga de ejecutar el algoritmo Pagerank en giraph y mapreduce por cada archivo que se encuentra el directorio grafos/ del hdfs.

```
#!/usr/bin/env bash
if [ -z "$1" ]; then
echo "Ingrese nro de workers"
exit
fi
```

```

workers=$1
#Recorremos los archivos del directorio grafos/ del
hdfs
for i in $(hadoop --loglevel FATAL fs -ls -C grafos/)
do
    echo $i - workers: $workers
    if [ "$i" == "grafos/web-wikipedia_link_it.txt" ]
    then
        iter=75
        nodes=2790239
    else
        iter=49
        nodes=4847571
    fi
    ./config.sh $nodes
    ./giraph-run-time.sh $workers $i $iter
    ./hadoop-run-time.sh $workers $i $iter
done

```

**giraph-run-time.sh:** Espera como parámetro el número de workers, el archivo a procesar y la cantidad de iteraciones que debe realizar.

Su función es ejecutar el programa SimplePageRankComputation de Giraph iterando la cantidad de veces indicada por parámetro.

Previo se ejecuta el mapreduce src/sortMapper.py y src/sortReducer.py que convierte el archivo de entrada al formato que espera Giraph.

También genera un log muy útil para analizar los resultados.

```

#!/usr/bin/env bash
workers=$1
input=$2
times=$3
input_file=$(basename "$input")
hostname=cluster-arellanon-$workers-m
output=/user/naheel/out-giraph-w-$workers-$input_file
log=~/.log-giraph-w-$workers-$input_file

hadoop fs -rm -r $output

echo ..... >>
$log
echo giraph - $input_file - worker $workers >>
$log
echo ..... >>
$log

hadoop fs -rm -r tmp input_tmp
hadoop fs -mkdir input_tmp

```

```

#hadoop fs -rm -r tmp_values
hadoop fs -cp $input input_tmp/

timeInicio=$(date +%s)

echo inicio: $(date) >> $log
hadoop jar /usr/lib/hadoop-mapreduce/hadoop-streaming.jar -D mapred.reduce.tasks=1 \
    -file src \
        -mapper "src/sortMapper.py" \
        -reducer "src/sortReducer.py" \
        -input input_tmp \
        -output tmp

hadoop jar /usr/local/giraph/giraph-examples/target/giraph-examples-with-dependencies.jar org.apache.giraph.GiraphRunner \
    -Dmapred.job.tracker=$hostname \
    -libjars /usr/local/giraph/giraph-examples/target/giraph-examples-with-dependencies.jar org.apache.giraph.examples.SimplePageRankComputation \
    -vif org.apache.giraph.io.formats.JsonLongDoubleFloatDoubleVertexInputFormat \
    -vip tmp/part-00000 \
    -vof org.apache.giraph.io.formats.IdWithValueTextOutputFormat \
    -op $output \
    -mc "org.apache.giraph.examples.SimplePageRankComputation\${SimplePageRankMasterCompute}" \
    -ca giraph.SplitMasterWorker=true \
    -ca giraph.maxNumberOfSupersteps=$times \
    -w $workers

timeFin=$(date +%s)
duracion=$((($timeFin-$timeInicio))
min=$(( $duracion/60))
seg=$(( $duracion-(min*60))

echo fin:      $(date) >> $log
echo duracion: $min:$seg >> $log
hadoop fs -rm -r input_tmp tmp tmp2

```

**hadoop-run-time.sh:** Espera como parámetro el número de workers, el archivo a procesar y la cantidad de iteraciones que debe realizar. Su función es ejecutar el mapreduce `src/prMapper.py` y `src/prReducer.py`. Estos scripts fueron desarrollados en Python para este proyecto.



También genera un log muy útil para analizar los resultados.

```
#!/usr/bin/env bash
if [ -z "$1" ]; then
echo "Ingrese nro de workers"
exit
fi
workers=$1
input=$2
times=$3
input_file=$(basename "$input")

output=/user/nahuel/out-hadoop-w-$workers-$input_file
log=~/.log-hadoop-w-$workers-$input_file

#Se genera archivo temporal de values
hadoop fs -rm -r $output
hadoop fs -mkdir $output
hadoop fs -rm -r input_tmp
hadoop fs -mkdir input_tmp

hadoop fs -cp $input input_tmp/

echo ..... >>
$log
echo hadoop - $input_file - worker $workers >>
$log
echo ..... >>
$log
totTimeInicio=$(date +%s)
for i in `seq 1 $times`
do
    echo iteracion: $i
    echo .....
.. >> $log
    echo interacion: $i - $input_file >> $log
    echo .....
.. >> $log
    hadoop fs -rm -r tmp tmp2 tmp3
    timeInicio=$(date +%s)
    echo $i - inicio: $(date) >> $log

    hadoop jar /usr/lib/hadoop-mapreduce/hadoop-streaming.jar \
        -file src \
            -mapper "src/sortMapper.py" \
            -reducer "src/sortReducer.py" \
```

```

        -input input_tmp \
        -output tmp

    hadoop jar /usr/lib/hadoop-mapreduce/hadoop-streaming.jar \
        -file src \
        -mapper "src/prMapper.py" \
        -reducer "src/prReducer.py" \
            -input tmp/part-* \
            -output tmp2

    #el output es la nueva entrada para la proxima
iteracion
    hadoop fs -rm -r input_tmp/part-*
    hadoop fs -mv tmp2/part-* input_tmp/

    timeFin=$(date +%s)
    duracion=$((($timeFin-$timeInicio))
    min=$((($duracion/60))
    seg=$((($duracion-(min*60))
    echo $i - fin:      $(date)    >> $log
    echo duracion: $min:$seg      >> $log
done
totTimeFin=$(date +%s)
totDuracion=$((($totTimeFin-$totTimeInicio))
totMin=$((($totDuracion/60))
totSeg=$((($totDuracion-(totMin*60))
echo Con $workers worker - duracion total: $tot-
Min:$totSeg    >> $log
hadoop fs -mv input_tmp/part-* $output
#hadoop fs -mv input_tmp/tmp_values $output
hadoop fs -rm -r input_tmp tmp tmp2 tm3

```

**hadoop-run-converge.sh:** Espera como parámetro el número de workers y el archivo a procesar. Este script es muy similar a `hadoop-runtime.sh`, pero en lugar de iterar por una cantidad de veces indicada por parámetro, el programa se ejecutará hasta que el pagerank converja por debajo de un umbral determinado, en este caso 0.1.

Para validar si el pagerank converge se ejecuta el mapreduce `src/sortMapper.py` Y `src/MaxDiff.py` después de cada iteración y verifica si está por debajo del umbral.

Este script es útil para determinar la cantidad de iteraciones que se deben realizar sobre los grafos de prueba.

```

#!/usr/bin/env bash
if [ -z "$1" ]; then
echo "Ingrese nro de workers"
exit
fi
workers=$1
input=$2
#times=$3
input_file=$(basename "$input")
CONTINUAR=1
umbral=0.1
i=1

output=/user/nahuel/out-hadoop-w-$workers-$input_file
log=~/.log-hadoop-w-$workers-$input_file

#Se genera archivo temporal de values
hadoop fs -rm -r $output
hadoop fs -mkdir $output
hadoop fs -rm -r input_tmp
hadoop fs -mkdir input_tmp

hadoop fs -cp $input input_tmp/

echo ..... >>
$log
echo hadoop - $input_file - worker $workers >>
$log
echo ..... >>
$log
totTimeInicio=$(date +%s)
#for i in `seq 1 $times`
while [ $CONTINUAR -eq 1 ]
do
    echo iteracion: $i
    echo .....
.. >> $log
    echo interacion: $i - $input_file >> $log
    echo .....
.. >> $log
    hadoop fs -rm -r tmp tmp2 tmp3
    timeInicio=$(date +%s)
    echo $i - inicio: $(date) >> $log

    hadoop jar /usr/lib/hadoop-mapreduce/hadoop-streaming.jar \
        -file src \
        -mapper "src/sortMapper.py" \

```

```

        -reducer "src/sortReducer.py" \
        -input input_tmp \
        -output tmp

    hadoop jar /usr/lib/hadoop-mapreduce/hadoop-streaming.jar \
        -file src \
        -mapper "src/prMapper.py" \
        -reducer "src/prReducer.py" \
            -input tmp/part-* \
            -output tmp2

    if [ $i -gt 1 ]; then
        hadoop jar /usr/lib/hadoop-mapreduce/hadoop-streaming.jar -D mapred.reduce.tasks=1 \
            -file src \
            -mapper "src/sortMapper.py" \
            -reducer "src/MaxDiff.py" \
                -input input_tmp/part-* \
                -input tmp2/part-* \
                -output tmp3

        delta=`hadoop fs -cat tmp3/part-00000`
        CONTINUAR=$( echo "$delta>$umbral" | bc )
        echo $delta
        echo $CONTINUAR
        hadoop fs -rm -r tmp3
    fi
    #el output es la nueva entrada para la proxima iteracion
    hadoop fs -rm -r input_tmp/part-*
    hadoop fs -mv tmp2/part-* input_tmp/

    timeFin=$(date +%s)
    duracion=$((($timeFin-$timeInicio))
    min=$((($duracion/60))
    seg=$((($duracion-(min*60))
    echo $i - fin:      $(date)  >> $log
    echo duracion: $min:$seg    >> $log

    i=$((i+1))
done
totTimeFin=$(date +%s)
totDuracion=$((($totTimeFin-$totTimeInicio))
totMin=$((($totDuracion/60))
totSeg=$((($totDuracion-(totMin*60))
echo Con $workers worker - duracion total: $totMin:$totSeg    >> $log

```

```
hadoop fs -mv input_tmp/part-* $output
hadoop fs -rm -r input_tmp tmp tmp2 tm3
```

- Se eligió un valor de umbral alto, porque se demoraba más de 12hs sobre la mejor infraestructura de clúster disponible.

## Implementación

La implementación del proyecto PageRank-Hadoop se divide en dos pasos, primero determinar la cantidad de iteraciones necesarias para cada grafo y después ejecutar las pruebas sobre los clusters.

### Primer paso:

Se debe ejecutar el script `hadoop-run-converge.sh` para determinar la cantidad de iteraciones necesarias para que los grafos converjan.

Para eso en `project_run.sh` se comenta el llamado a los bash `./giraph-run-time.sh`

Y `./hadoop-run-time.sh` y se agrega la línea `./hadoop-run-converge.sh`

`project_run.sh`

```
#!/usr/bin/env bash
if [ -z "$1" ]; then
echo "Ingrese nro de workers"
exit
fi
workers=$1
#Recorremos los archivos del directorio grafos/ del
hdfs
for i in $(hadoop --loglevel FATAL fs -ls -C grafos/)
do
    echo $i - workers: $workers
    if [ "$i" == "grafos/web-wikipedia_link_it.txt" ]
    then
#         iter=?
        nodes=2790239
    else
#         iter=?
        nodes=4847571
    fi
    ./config.sh $nodes
#     ./giraph-run-time.sh $workers $i $iter
```

```
# ./hadoop-run-time.sh $workers $i $iter
  ./hadoop-run-converge.sh $workers $i
done
```

Luego se procede a ejecutar el script `./install.sh workers`  
 Una vez finalizada la ejecución se analiza el log y se determina la cantidad de iteraciones necesaria para cada grafo.

Con un umbral de 0.1 devuelve los siguientes resultados:

	Alias	Iteraciones
<b>Wikipedia</b>	<b>D1</b>	<b>75</b>
<b>Social Live journal</b>	<b>D2</b>	<b>49</b>

*Tabla 3*

Se utilizó un valor de umbral alto por los tiempos de procesamiento que conllevaban las pruebas, con umbrales más bajo los experimentos hubiesen consumido más tiempo de procesamiento y este es un recurso limitado.

### Segundo paso:

Con esta información se procede a ejecutar las pruebas sobre los cluster con 0, 2, 4 y 6 workers. Se configura el archivo `project_run.sh` con las iteraciones correspondientes para cada archivo:

```
project_run.sh

#!/usr/bin/env bash
if [ -z "$1" ]; then
echo "Ingrese nro de workers"
exit
fi
workers=$1
#Recorremos los archivos del directorio grafos/ del
hdfs
for i in $(hadoop --loglevel FATAL fs -ls -C grafos/)
do
    echo $i - workers: $workers
    if [ "$i" == "grafos/web-wikipedia_link_it.txt" ]
    then
        iter=75
        nodes=2790239
```

```

else
    iter=49
    nodes=4847571
fi
./config.sh $nodes
./giraph-run-time.sh $workers $i $iter
./hadoop-run-time.sh $workers $i $iter
#    ./hadoop-run-converge.sh $workers $i
done

```

Se establece conexión remota con el nodo master de cada cluster, se descarga el proyecto y se inicia una sesión virtual con el comando `screen`

```

gcloud compute ssh cluster-arellanon-workers-m
nahuel@cluster-arellanon-workers-m:~$ git clone
https://github.com/arellanon/PageRank-Hadoop
nahuel@cluster-arellanon-workers-m:~$ screen

```

Luego se procede a ejecutar el proyecto PageRank-Hadoop sobre los clusters:

```

nahuel@cluster-arellanon-0-m:~/PageRank-Hadoop$ ./install.sh 1
nahuel@cluster-arellanon-2-m:~/PageRank-Hadoop$ ./install.sh 2
nahuel@cluster-arellanon-4-m:~/PageRank-Hadoop$ ./install.sh 4
nahuel@cluster-arellanon-6-m:~/PageRank-Hadoop$ ./install.sh 6

```

## MÉTRICAS

### Tiempos de procesamiento

Para el proceso de ejecución de PageRank la métrica más importante suele ser el tiempo de construcción del ranking.

A continuación, se presenta las tablas Tabla 4 - MapReduce y Tabla 5 - Giraph con los tiempos de ejecución en unidad de minutos del algoritmo PageRank para los datasets D1 y D2.

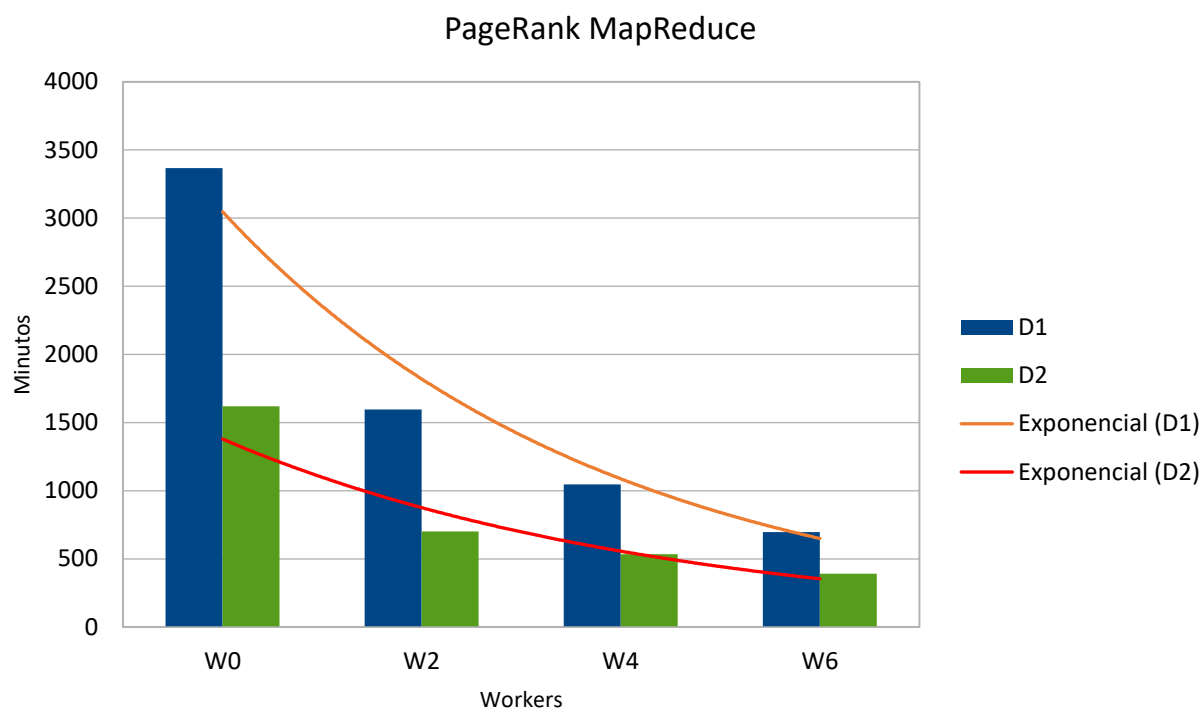
WORKERS	D1	D2
W0	3368,06	1620,40
W2	1596,58	701,55
W4	1045,48	533,71
W6	696,38	391,10

*Tabla 4 - MapReduce*

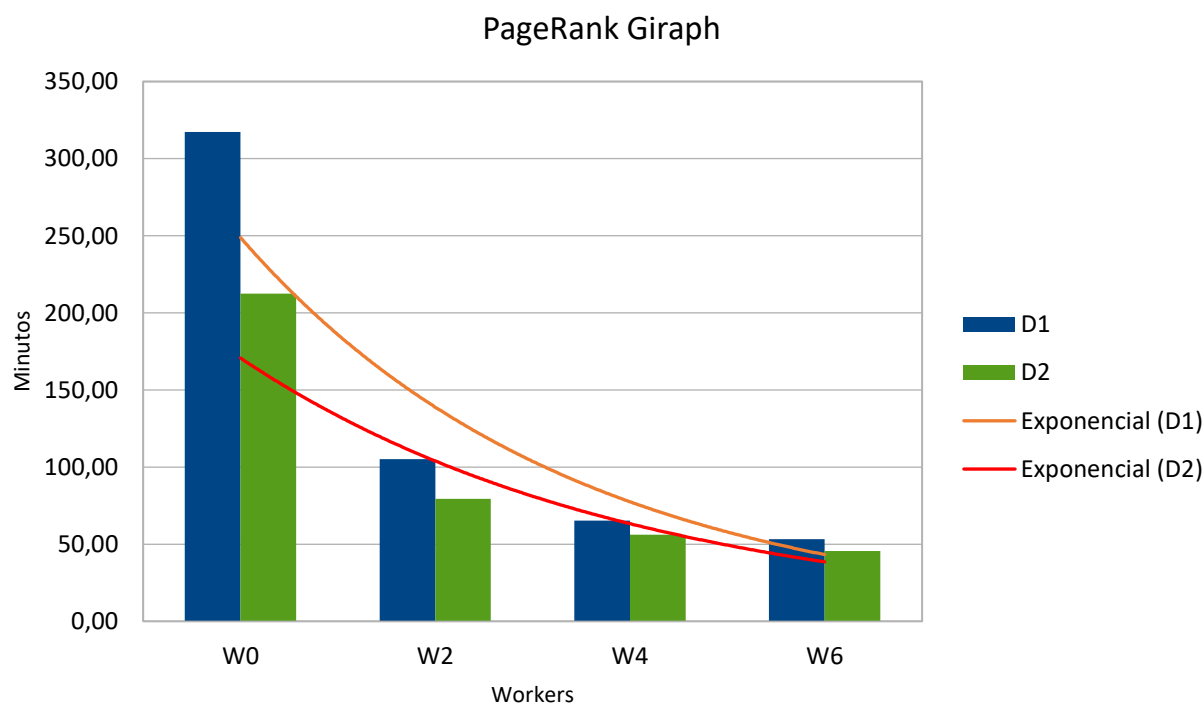
WORKERS	D1	D2
W0	317,20	212,43
W2	105,20	79,50
W4	65,21	56,13
W6	53,25	45,66

*Tabla 5 - Giraph*





*Ilustración 4*



*Ilustración 5*

En la Tabla 4 - MapReduce y Tabla 5 - Giraph al igual que en las figuras Ilustración 4 e Ilustración 5, se observan que en la medida que se incrementan la cantidad de nodos se advierte una tendencia al descenso en los tiempos. Este comportamiento general se repite en MapReduce y Giraph.

## Speedup

El Speedup es una medida de la mejora de rendimiento de la aplicación al aumentar la cantidad de procesadores (comparado con el rendimiento al utilizar un solo procesador). Formalmente se define como:

$$S_p = \frac{T_1}{T_N}$$

Donde:

**Sp** es el Speedup del algoritmo ejecutándose en n núcleos

**T1** es el tiempo de ejecución del algoritmo en 1 núcleo o de manera secuencial

**Tn** es el tiempo de ejecución del algoritmo en n núcleos

A partir de los tiempos analizados, se puede construir una curva de Speedup para determinar cuánto se mejora el rendimiento de los algoritmos con la incorporación de nodos.

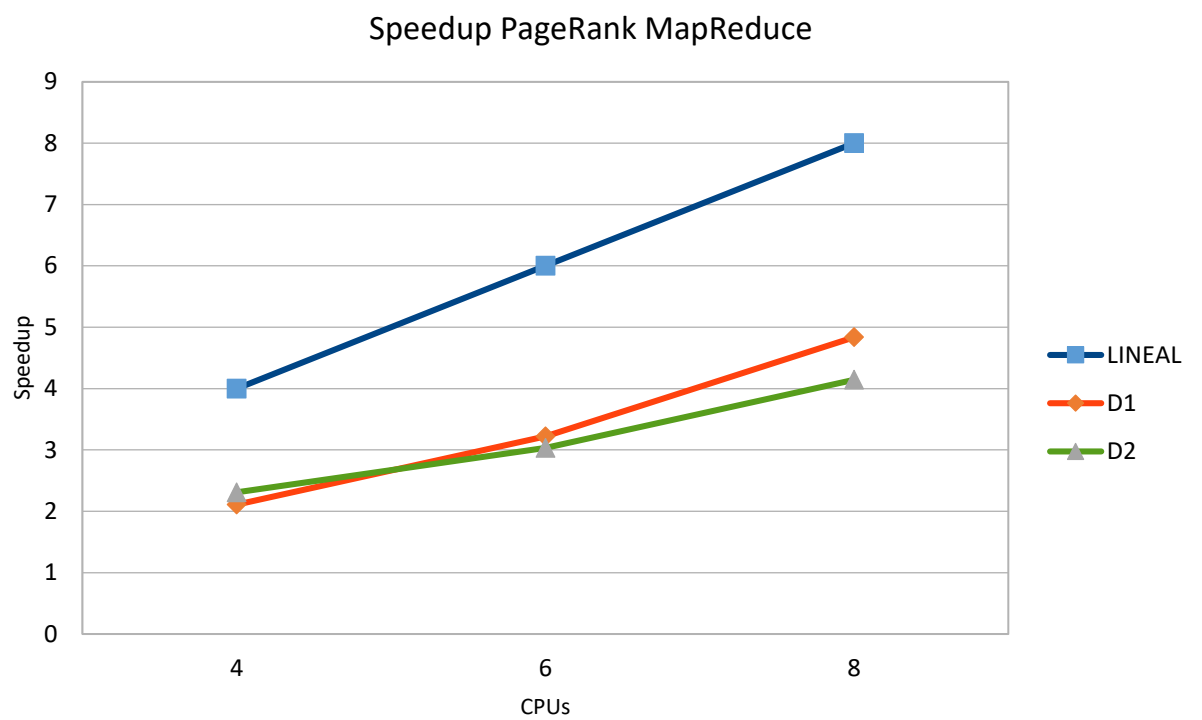
En las tablas Tabla 6 - MapReduce y Tabla 7 - Giraph se visualiza el speedup por cada dataset y cantidad de cpus utilizados en la ejecución del algoritmo.

CPUs	LINEAL	D1	D2
4	4	2,11	2,31
6	6	3,22	3,04
8	8	4,84	4,14

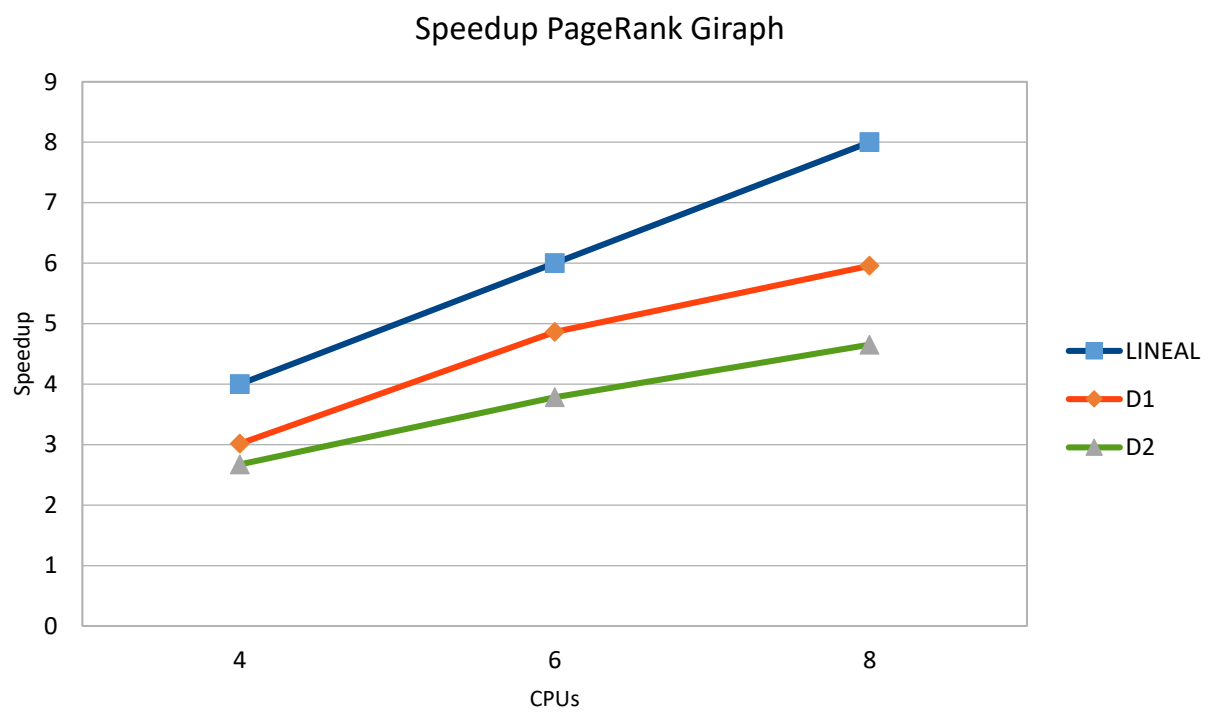
*Tabla 6 - MapReduce*

CPUs	LINEAL	D1	D2
4	4	3,02	2,67
6	6	4,86	3,78
8	8	5,96	4,65

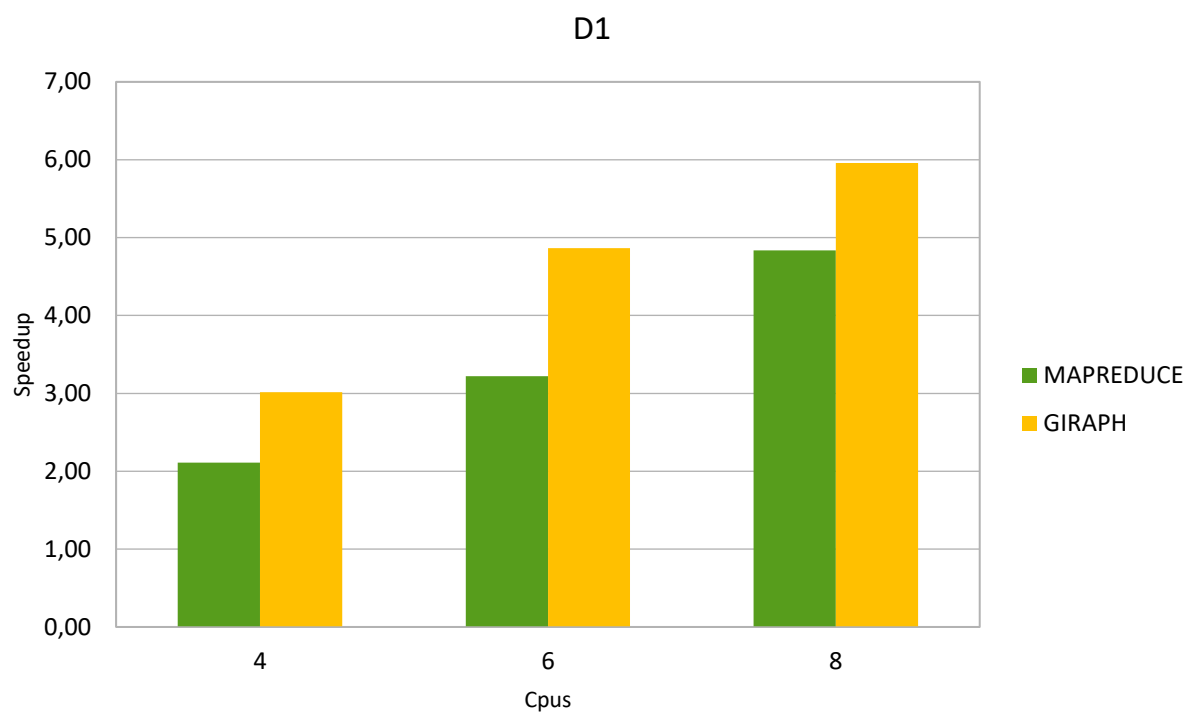
*Tabla 7 - Giraph*



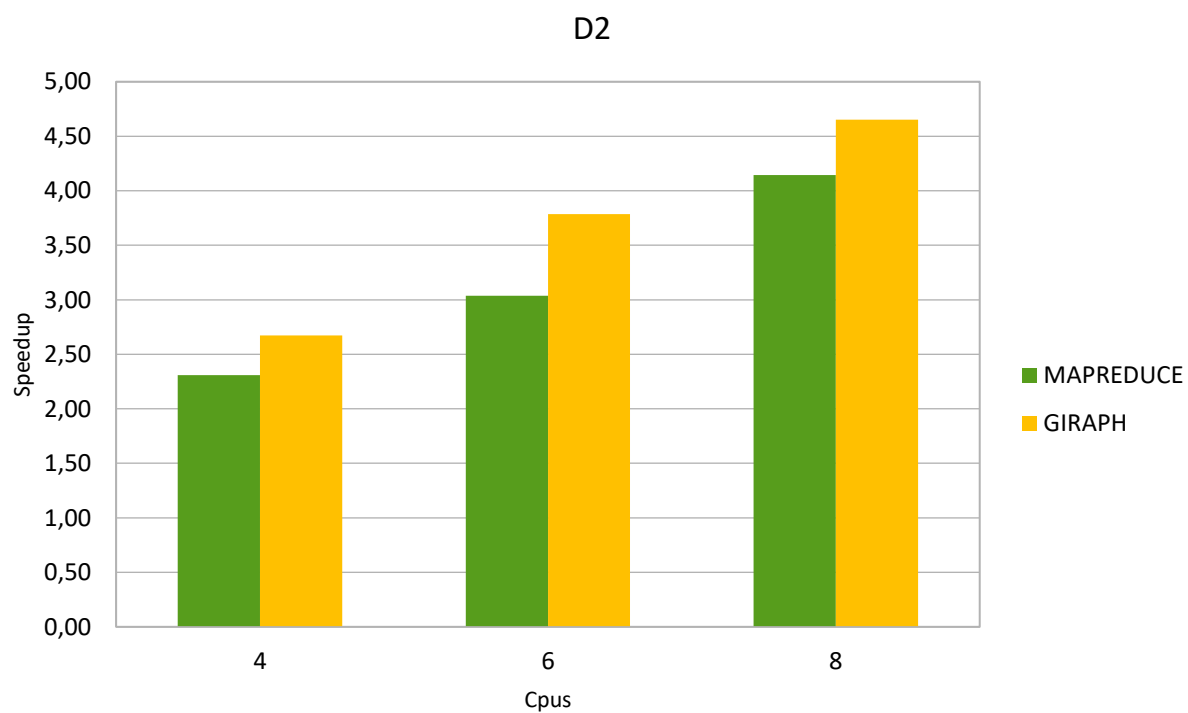
*Ilustración 6*



*Ilustración 7*



*Ilustración 8*



*Ilustración 9*

Como se puede visualizar en la Ilustración 6 e Ilustración 7 las diferencias entre las curvas de Speedup respecto de la curva lineal indica que siempre es posible realizar mejoras al algoritmo, aunque debe tenerse en cuenta la sobrecarga de procesamiento propia de la plataforma.

También se puede observar en las figuras Ilustración 8 e Ilustración 9 que Giraph realiza una mejor escalabilidad que MapReduce sobre los dataset D1 y D2 para todas las configuraciones de cluster donde se realizaron las pruebas.

Por su parte, la eficiencia cuantifica el speedup obtenido por procesador y se define como  $E(n) = S_n/n$  donde  $S_n$  es el speedup utilizando  $n$  procesadores.

La eficiencia permite medir de forma sencilla como se están utilizando los recursos. La eficiencia puede ayudar a comprender que, aunque se inviertan recursos en cantidad y se logre mejorar el Speedup, los mismos pueden ser excesivos y se estén desperdiciando en relación a los resultados obtenidos.

A continuación, se presentan las tablas de eficiencia:

CPUs	D1	D2
4	0,53	0,58
6	0,54	0,51
8	0,60	0,52

*Tabla 8 - MapReduce*

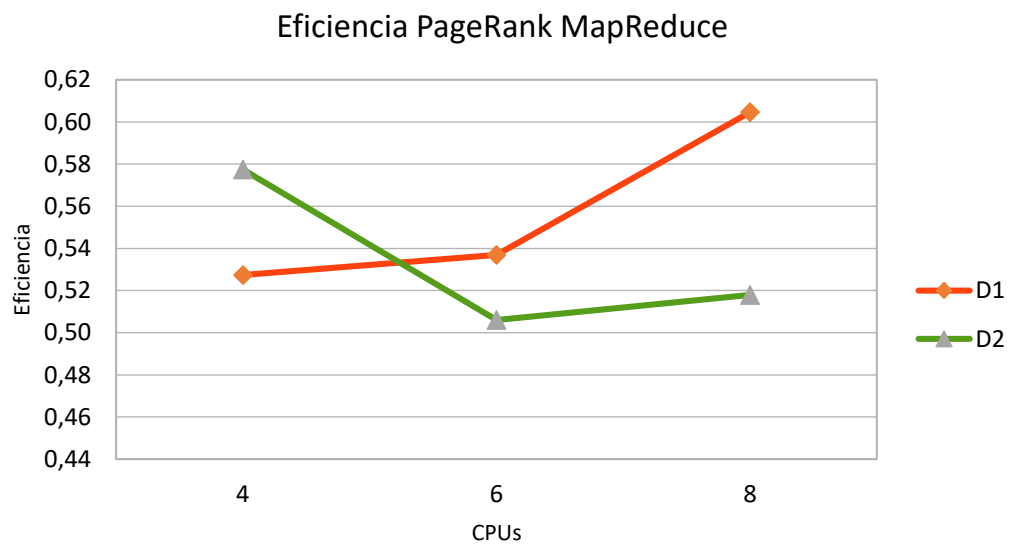
CPUs	D1	D2
4	0,75	0,67
6	0,81	0,63
8	0,74	0,58

*Tabla 9 - Giraph*

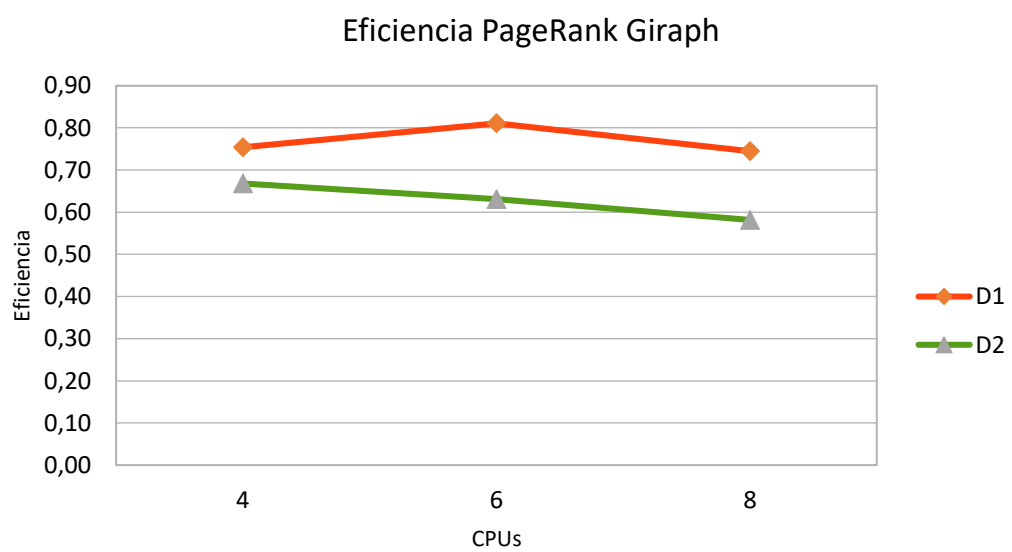
En las tablas Tabla 8 - MapReduce y Tabla 9 - Giraph se puede visualizar que para el dataset D1 el incremento de los cpus mejora la eficiencia, salvo en el caso de la ejecución con 8 cpus de Giraph.

En contraste, para el dataset D2 se observa que la eficiencia disminuye al aumentar la cantidad de cpus.

Se plantea como hipótesis para futuras investigaciones, que este decrecimiento de la eficiencia cuando se escala horizontalmente sobre el dataset D2, es por el tamaño de la muestra, es significativamente más pequeño que el D1 (representa el 73%) y por ende no puede hacer un uso más eficiente de plataformas con más cpus.



*Ilustración 10*



*Ilustración 11*

## MAPREDUCE VS GIRAPH

Con toda la información recopilada de las pruebas se puede hacer una comparación entre las plataformas MapReduce y Giraph.

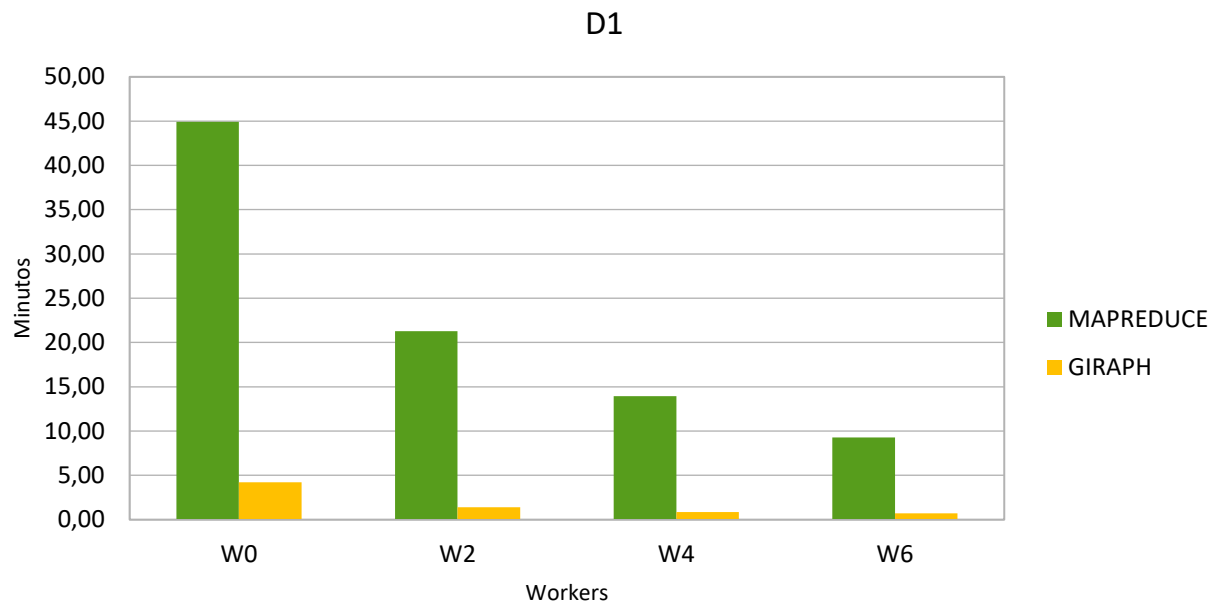
Se realiza una aproximación estableciendo un tiempo promedio por iteración entre el tiempo total y la cantidad de iteraciones por cada prueba.

ITERACIONES	75			
D1	MAPREDUCE		GIRAPH	
	TOTAL	PROMEDIO POR ITERACION	TOTAL	PROMEDIO POR ITERACION
W0	3368,06	44,91	317,20	4,23
W2	1596,58	21,29	105,20	1,40
W4	1045,48	13,94	65,21	0,87
W6	696,38	9,29	53,25	0,71

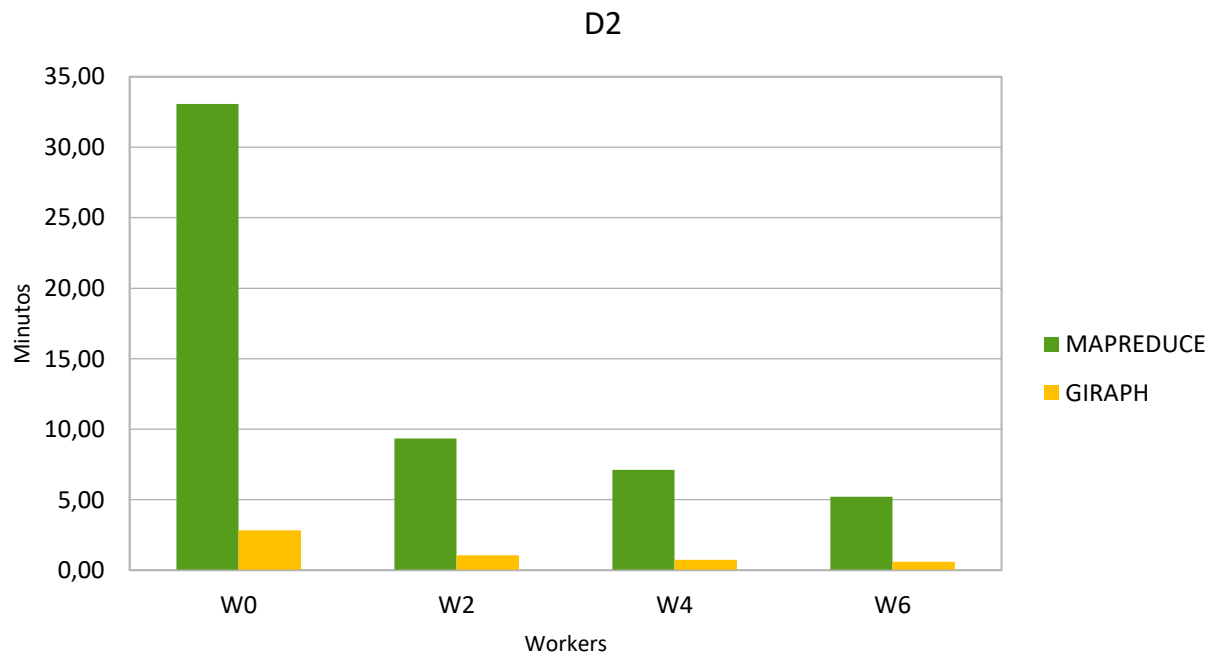
Tabla 10

ITERACIONES	49			
D2	MAPREDUCE		GIRAPH	
	TOTAL	PROMEDIO POR ITERACION	TOTAL	PROMEDIO POR ITERACION
W0	1620,40	33,07	212,43	2,83
W2	701,55	9,35	79,50	1,06
W4	533,71	7,12	56,13	0,75
W6	391,10	5,21	45,66	0,61

Tabla 11



*Ilustración 12*



*Ilustración 13*

Como se puede observar en las tablas Tabla 10 y Tabla 11, y sus respectivas ilustraciones Ilustración 12 e Ilustración 13 en los resultados Giraph tiene un rendimiento de 13 veces superior sobre Mapreduce para el dataset D1, y para el datase D2 de casi 10.



## CONCLUSIONES

El presente trabajo es fruto del esfuerzo de combinar técnicas de clasificación de páginas web con tecnologías de Big Data. La motivación principal se centró en el desafío de construir y analizar el comportamiento del algoritmo Pagerank en un entorno distribuido donde se permita escalar y estudiar su comportamiento.

Se desarrolló e implementó el algoritmo Pagerank en MapReduce y se implementó en Giraph para la clasificación de páginas web según su relevancia.

También se construyó un grupo de cluster utilizando un proveedor en la nube y se pudo constatar la robustez de la plataforma Hadoop en el uso intensivo de la misma.

Otras conclusiones obtenidas a raíz del trabajo fueron:

- De acuerdo a los resultados obtenidos en las métricas de tiempo de procesamiento, speedup y eficiencia, Giraph presenta un rendimiento muy superior sobre MapReduce, esto se debe a que MapReduce realiza muchas tareas encadenadas hasta lograr su objetivo, y cada tarea implica entrada y salida a disco, en su lugar Giraph al poder mantener una comunicación global entre cada superstep sin requerir entradas y salidas a disco obtiene un mejor rendimiento.
- Respecto a la eficiencia, se observó que existen casos que la métrica no mejora a medida que se escala horizontalmente, se plantea como hipótesis para futuras investigaciones, que el motivo se debe al tamaño muestra, se necesita conjunto de datos de mayor tamaño para verificar un mejor rendimiento en la eficiencia a medida que se incrementa el tamaño del cluster.
- Cabe mencionar que, durante los experimentos, Giraph requería una mejor arquitectura de hardware para procesar el mismo conjunto de datos que con Mapreduce. También, se observó que la documentación es muy escasa y presenta complicaciones en su instalación y configuración en el cluster.

## Referencias

- [1] «IBM,» [En línea]. Available: <https://www.ibm.com/developerworks/ssa/local/im/que-es-big-data/index.html>.
- [2] «Emc,» [En línea]. Available: <https://www.emc.com/leadership/digital-universe/2014iview/executive-summary.htm>.
- [3] «Gartner,» [En línea]. Available: <https://www.gartner.com/it-glossary/big-data>.
- [4] «Apache Hadoop,» [En línea]. Available: <http://hadoop.apache.org/index.pdf>.
- [5] J. y. S. G. Dean, «MapReduce: Simplified Data Processing on Large Clusters».
- [6] F. M. y. F. M. Renzo Angles, «Supporting Property Graphs in Apache Giraph».
- [7] A. N. Langville y C. D. Meyer, Google's PageRank and Beyond: The Science of Search Engine Rankings, Princeton University Press., 2006.
- [8] M. Richardson y P. Domingos, The intelligent surfer: Probabilistic combination of link and content information in PageRank, 2002.
- [9] «Stanford,» [En línea]. Available: <https://snap.stanford.edu/data/wikispeedia.html>.
- [10] «Stanford,» [En línea]. Available: <https://snap.stanford.edu/data/soc-LiveJournal1.html>.
- [11] S. Wasserman y K. Faust, Analisis de redes sociales. Métodos y aplicaciones, 2013.

Desarrollo publicado: <https://github.com/arellanon/PageRank-Hadoop>