ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

FACULTAD DE INGENIERÍA EN ELECTRICIDAD Y COMPUTACIÓN

INFORME DE MATERIA DE GRADUACIÓN

"EVALUACIÓN, ANÁLISIS Y COMPARACIÓN DEL RENDIMIENTO DE PROGRAMAS DE PROCESAMIENTO MASIVO IMPLEMENTADOS USANDO LENGUAJES DE PROGRAMACIÓN JAVA, PYTHON Y C++ SOBRE LA PLATAFORMA HADOOP PARA CLÚSTERES DE VARIOS TAMAÑOS"

Previo a la obtención del título de:

INGENIERO EN CIENCIAS COMPUTACIONALES
ESPECIALIZACIÓN SISTEMAS TECNOLÓGICOS

PRESENTADA POR:

MAYRA ALEJANDRA MENDOZA SALTOS
BETSY TATIANA TRUJILLO MIRANDA

GUAYAQUIL – ECUADOR

2010

AGRADECIMIENTO

A Dios.

A nuestros padres, familiares y amigos
por el apoyo y las palabras de aliento
en los momentos difíciles.

Especialmente a MsC. Cristina Abad
y a la MsC. Ana Tapia
por su incondicional
apoyo como mentoras y consejeras
en todo momento.

DEDICATORIA

A Dios.
A nuestros padres por su cariño,
apoyo incondicional,
por toda la fe y esperanza
depositada en nosotras.

A nuestros familiares y amigos.

TRIBUNAL DE SUSTENTACIÓN

PROFESORA DE LA MATERIA DE GRADUACIÓN

MsC. Cristina Abad R.

PROFESORA DELEGADA POR EL DECANO

MsC. Ana Tapia

DECLARACIÓN EXPRESA

“La responsabilidad por los hechos, ideas y doctrinas expuestas en esta tesis, me corresponden exclusivamente; y, el patrimonio intelectual de la misma, a la Escuela Superior Politécnica del Litoral”

(Reglamento de exámenes y títulos profesionales de la ESPOL)

Mayra Alejandra Mendoza Saltos	Betsy Tatiana Trujillo Miranda

RESUMEN

En el presente trabajo se exhibe el análisis del rendimiento de lenguajes de programación tales como Java, C++ y Python sobre la plataforma Hadoop. Para su evaluación se han implementado cuatro programas típicos de procesamiento masivo de datos. El documento se ha dividido en cuatro capítulos que comprende desde fundamento teórico hasta las soluciones y los resultados obtenidos.

En el Capítulo 1 se describe la necesidad de obtener un criterio de partida al momento de elegir un lenguaje de programación para resolver una tarea específica, se detallan los objetivos planteados y el alcance del presente trabajo.

En el Capítulo 2 se introduce el concepto de Hadoop como plataforma de procesamiento masivo de datos y los complementos empleados tales como Streaming y Pipes.

En el Capítulo 3 se detallan los problemas planteados, las soluciones dadas, así como los formatos de entrada/salida y las librerías usadas.

Finalmente en el Capítulo 4 se detallan los resultados obtenidos y se realiza la comparación y el análisis de los mismos.

ÍNDICE GENERAL

ÍNDICE DE GRÁFICOS

ABREVIATURAS

HDFS	Hadoop File System
GFS	Google File System
API	Interfaz de programación de aplicaciones
STDIN	Entrada estándar
STDOUT	Salida estándar
STL	Standard Template Library
JNI	Java Native Interface
FIEC	Facultad de Ingeniería Electrónica y Computación
ESPOL	Escuela Superior Politécnica del Litoral
JAI	Java Advance Image
PIL	Python Image Library
JPG	Joint Photographic (Experts) Group
TI	Tecnología de Información
GB	Gigabyte
XML	Extensible Markup Language
EC2	Elastic Computing Cloud
S3	Simple Storage Service
MR	MapReduce

INTRODUCCIÓN

Al momento de desarrollar una aplicación es de vital importancia decidir qué lenguaje de programación se empleará. Generalmente se consideran factores determinantes como facilidad de uso, simplicidad, rendimiento y portabilidad; además es necesario considerar factores externos que influyen en la elaboración de un proyecto como tiempo, personal disponible, preferencia de lenguajes de programación, entre otros.

En este proyecto, se decidió evaluar el rendimiento de tres lenguajes de programación de uso común: Java, C++ y Python sobre la plataforma Hadoop.

En el presente trabajo se muestran los resultados obtenidos así como las conclusiones que le servirán de guía al momento de tomar una decisión.

CAPÍTULO 1

1. ANTECEDENTES Y OBJETIVOS

1.1. Antecedentes

En la actualidad existen estudios que evalúan el rendimiento de Hadoop bajo diferentes circunstancias. Uno de estos estudios es el paper A Comparison of Approaches to Large-Scale Data Analysis [1], en este trabajo se describen y comparan el paradigma MapReduce (MR) y SQL un Sistema de Gestión de Base de Datos (DBMS) y por otra parte, también evalúan los dos tipos de sistemas en términos de rendimiento y desarrollo, los resultados revelan algunas interesantes ventajas y desventajas.

Otro estudio realizado es Estimating Language Models Using Hadoop and Hbase [2], esta tesis presenta el trabajo de construcción a gran escala del modelo distribuido en el lenguaje ngram utilizando la plataforma MapReduce de Hadoop y una base de datos distribuida llamada Hbase. Donde proponen un método centrado en el costo del tiempo y el tamaño de almacenamiento del modelo, la exploración de diferentes estructuras de la tabla Hbase y enfoques de compresión. El método se aplica a construir procesos de formación y las pruebas utilizando

MapReduce y Hbase. Las pruebas de evaluar y comparar diferentes estructuras de tabla de la formación de 100 millones de modelos de palabras en unigramas, bigramas y trigramas, y los resultados sugieren una tabla basada en la estructura media ngram es una buena opción para el lenguaje de modelo distribuido.

El estudio realizado en el presente proyecto es diferente a los estudios ya existentes por lo que sirve de complemento en el análisis y comparación del rendimiento de Hadoop.

1.2. Objetivos

El objetivo general del presente trabajo es realizar una comparativa del rendimiento de programas desarrollados usando lenguajes de programación Java, C++ y Python sobre la plataforma Hadoop.

Los objetivos específicos son los siguientes:

- Implementar el programa WordCounter en Java, Python y C++.
- Implementar el programa Bi-gramas en Java, Python y C++.
- Implementar el programa Escala de Grises de imágenes en Java, Python y C++.
- Implementar el programa Hit Log FIEC-ESPOL en Java, Python y C++.

- Obtener tiempos de respuestas para cada aplicación ejecutada en 2, 4, 6, 10, 15 y 20 nodos.
- Realizar evaluación y comparación de los resultados obtenidos.
- Elaborar gráficas comparativas del tiempo de respuesta de cada aplicación.

CAPÍTULO 2

2. FUNDAMENTACIÓN TEÓRICA

2.1. Hadoop: Plataforma de procesamiento masivo de datos

El análisis masivo de datos se fundamenta en algo más que el proceso paralelo. También es necesario que el tratamiento de datos se haga con modelos reductores, ya que mientras menor información se procese, mejor es el rendimiento computacional.

2.1.1. Hadoop

Hadoop es un proyecto de Apache Software Foundation, subproyecto de Lucene [3], es una librería de software utilizada como base para muchos proyectos de recuperación de la información (information retrieval).

Es una plataforma para el procesamiento distribuido y masivo de datos, muy útil para realizar proyectos que necesiten de escalabilidad debido a que puede almacenar y procesar gran cantidad de datos del orden de los petabytes.

Hadoop se basa en el modelo de programación Map/Reduce [4], que permite dividir las aplicaciones en pequeñas tareas y procesarlas en paralelo en varias computadoras con el objetivo de reducir los tiempos de procesamiento de los datos. Se presenta como una solución de código abierto para los programadores sin experiencia en desarrollo de aplicaciones para ambientes distribuidos, ya que oculta la implementación de detalles propios de estos sistemas: paralelismo de tareas, tolerancia a fallos, administración de procesos y balanceo de carga.

Hadoop trabaja separando el procesamiento en dos fases: la fase map y la fase reduce. Cada fase tiene un par clave-valor como entrada y salida. El programador también especifica dos funciones: la función map y la función reduce. El lenguaje nativo de Hadoop es Java, la Figura 2.1.1.1 Flujo de datos lógico de MapReduce (tomado del libro Hadoop The Definitive Guide[15])explica detalladamente lo que sucede a lo largo del procesamiento MapReduce.

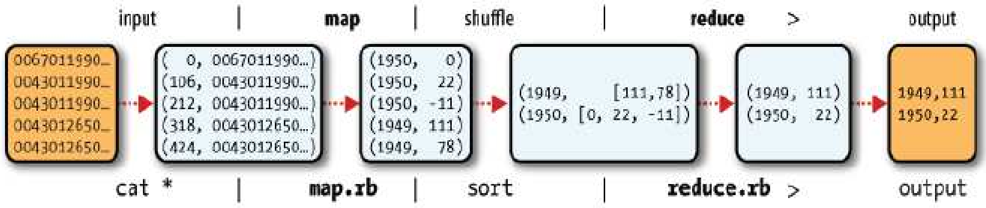


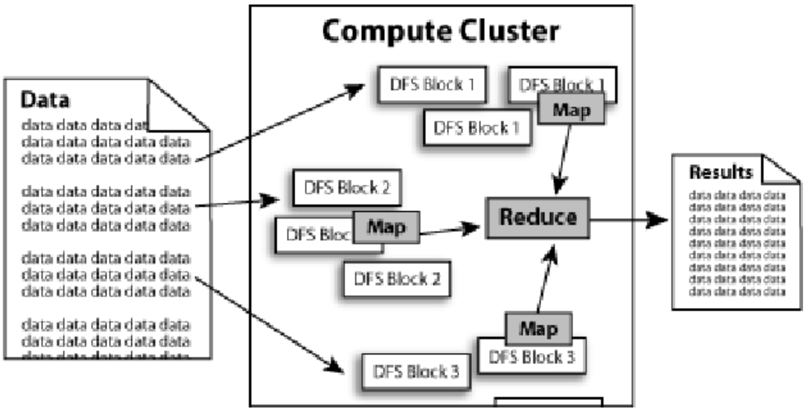
Figura 2.1.1.1 Flujo de datos lógico de MapReduce (tomado del libro Hadoop The Definitive Guide[15])

Hadoop usa como sistema de archivos el HDFS (Hadoop Distributed FileSystem) [5] el cual está basado en el sistema de archivos GFS [6] diseñado por Google.

El HDFS es un sistema de archivos distribuido diseñado para ejecutarse en hardware comercial. Tiene muchas similitudes con los actuales sistemas de archivos distribuidos. Sin embargo, las diferencias con otros sistemas de archivos distribuidos son significativas.

HDFS es altamente tolerante a fallos y está diseñado para ser implementado en el hardware de bajo costo, proporciona un acceso de alto rendimiento para aplicaciones de datos y es adecuado para aplicaciones que tienen grandes conjuntos de datos. Fue construido originalmente como la infraestructura de procesamiento distribuido para el motor de búsquedas Apache Nutch.

HDFS es parte del proyecto Hadoop Apache Core[7].



2.1.2. Streaming

Hadoop Streaming es un API genérica que viene con las distribuciones de Hadoop. Streaming permite que programas mapper/reducer escritos en cualquier lenguaje (ejecutables y scripts) puedan ser ejecutados sobre Hadoop.

Tanto el mapper como el reducer reciben su entrada en stdin (entrada estándar) dejándola lista para emitir la salida stdout(salida estándar) como un par clave / valor.

De forma predeterminada, un registro está dado por una línea, donde la primera parte (definida por el carácter de tabulación) es la Clave y el resto de la línea sin incluir el carácter de tabulación es el Valor.

Las entradas para el reducer se ordenan de forma que mientras cada línea contiene un único par clave/valor, todos los valores de la misma clave son adyacentes entre sí.

Siempre que se pueda manejar los datos de entrada en el formato de texto descrito anteriormente, cualquier programa de Linux o herramienta puede ser utilizada como el mapper/reducer en streaming.

Se puede escribir cualquier scripts en bash, python, perl o cualquier otro lenguaje, la condición es que el intérprete necesario está presente en todos los nodos del clúster.

Para este proyecto se hizo uso de Streaming para la ejecución de programas desarrollados en Python.

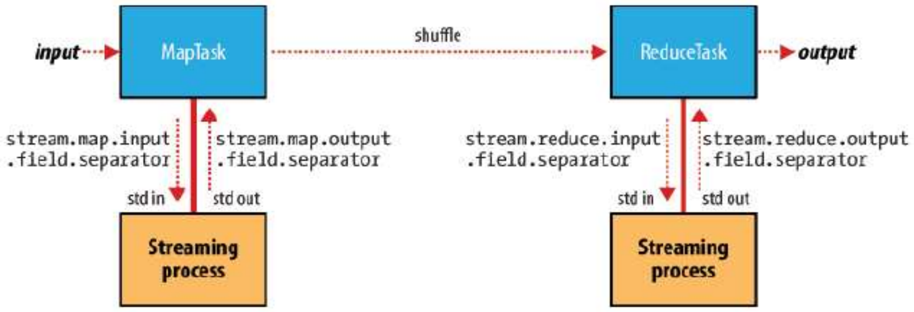


Figura 2.1.2.2 Donde los separadores son usados en un trabajo MapReduce Streaming (tomado del Libro Hadoop The Definitive Guide[14])

2.1.3. Pipes

Hadoop Pipes es el nombre de la interfaz de C++ para Hadoop MapReduce. Los archivos de inclusión y bibliotecas estáticas están presentes en C++/Linux-i386-32/ bajo el directorio de instalación de Hadoop.

A diferencia de la interfaz de Java, las entradas clave/valor para un programa que usa pipes son del tipo byte buffers representadas como cadenas de STL (Standard Template Library), esto hace la interfaz más simple aunque tiene mayor carga al momento de desarrollar las aplicaciones (haciendo una conversión a los datos para que puedan ser interpretados).

Streaming utiliza la entrada y salida estándar para comunicarse con el mapper y el reducer en su lugar Pipes usa sockets como canal de comunicación en el proceso de ejecución de funciones mapper/reducer Pipes no utiliza JNI.

Pipes no se ejecuta independiente, ya que depende del caché distribuido de Hadoop que sólo funciona cuando el HDFS se está ejecutando.

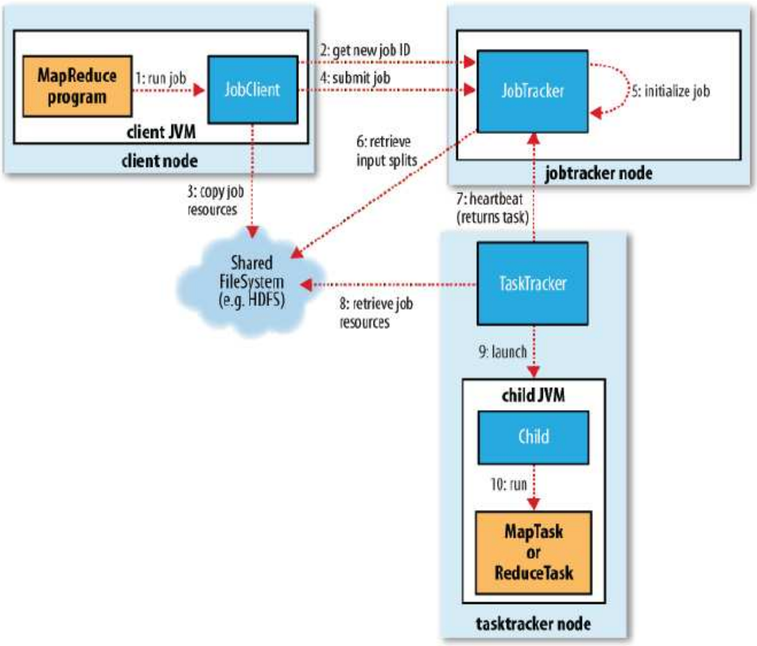
2.2. Ejecución de un trabajo MapReduce en Hadoop[8]

Si bien se puede ejecutar un trabajo MapReduce con una sola línea de código: JobClient.runJob(conf), esta corta línea esconde una gran cantidad de procesamiento. Esta sección describe el procesamiento involucrado para ejecutar un trabajo MapReduce en Java nativo, e indica las diferencias del proceso cuando se utiliza Streaming y cuando se utiliza Pipes. Comprender estas diferencias es crucial para entender los resultados de la posterior evaluación de estos tres mecanismos de ejecución de trabajos MapReduce en Hadoop.

Todo el proceso se ilustra en la

Figura 2.2.3 Cómo Hadoop ejecuta un trabajo MapReduce. En el nivel superior, hay cuatro entidades independientes:

- El cliente, que envía el trabajo MapReduce.
- El jobtracker, que coordina la ejecución del trabajo. El jobtracker es una aplicación en Java, cuya principal clase es JobTracker.
- El tasktrackers, que ejecutan las tareas en que se ha dividido el trabajo: Tasktrackers son aplicaciones en Java cuya principal clase es TaskTracker.
- El sistema de archivos distribuido, que se utiliza para archivos de



trabajo compartido entre las otras entidades.

Figura 2.2.3 Cómo Hadoop ejecuta un trabajo MapReduce(tomado del Libro Hadoop The Definitive Guide[21])

2.2.1. Envío de Trabajos

El método runJob() en JobClient es un método que crea una nueva instancia JobClient y llama al submitJob() en el mismo (paso 1 en la Figura 2.2.1). Cuando el trabajo está completo, si se ha realizado correctamente, los contadores de Trabajo se muestran. De lo contrario, el error que causó el trabajo no se registra en la consola.

2.2.2. Empleo de inicialización

Cuando el JobTracker recibe una llamada a su método submitJob(), se coloca en una cola interna desde donde el planificador de tareas lo recogerá y lo inicializará. La inicialización implica la creación de un objeto para representar el trabajo que está siendo ejecutado, que encapsula sus funciones.

2.2.3. Asignación de Tareas

Tasktrackers ejecutan un lazo simple que periódicamente envía método de las llamadas como latidos del corazón al jobtracker. Los latidos del corazón le dicen al jobtracker que un tasktracker está vivo, pero también sirven como un canal para los mensajes. Como parte de los latidos del corazón, un tasktracker indicará si está dispuesto a ejecutar una nueva tarea, y si lo está, el jobtracker asignará una tarea, que se comunica con el tasktracker utilizando el valor de retorno latidos del corazón (paso 7).

2.2.4. Ejecución de la Tarea

Ahora el tasktracker se le ha asignado una tarea, el siguiente paso es para que se ejecute la tarea. Primero, localiza el trabajo en JAR copiándolo del sistema de archivos compartidos al tasktracker de sistema de archivos. También copia los archivos necesarios de la memoria caché distribuida por la aplicación en el disco local (paso 8). Segundo, crea un directorio de trabajo local para la tarea, y se extrae el contenido del JAR en este directorio. Finalmente, crea una instancia de TaskRunner para ejecutar la tarea.

2.2.5. Streaming y Pipes

Ambos Streaming y Pipes ejecutan tareas especiales de map y reduce con el fin de entender el ejecutable suministrado por el usuario y comunicarse con él.

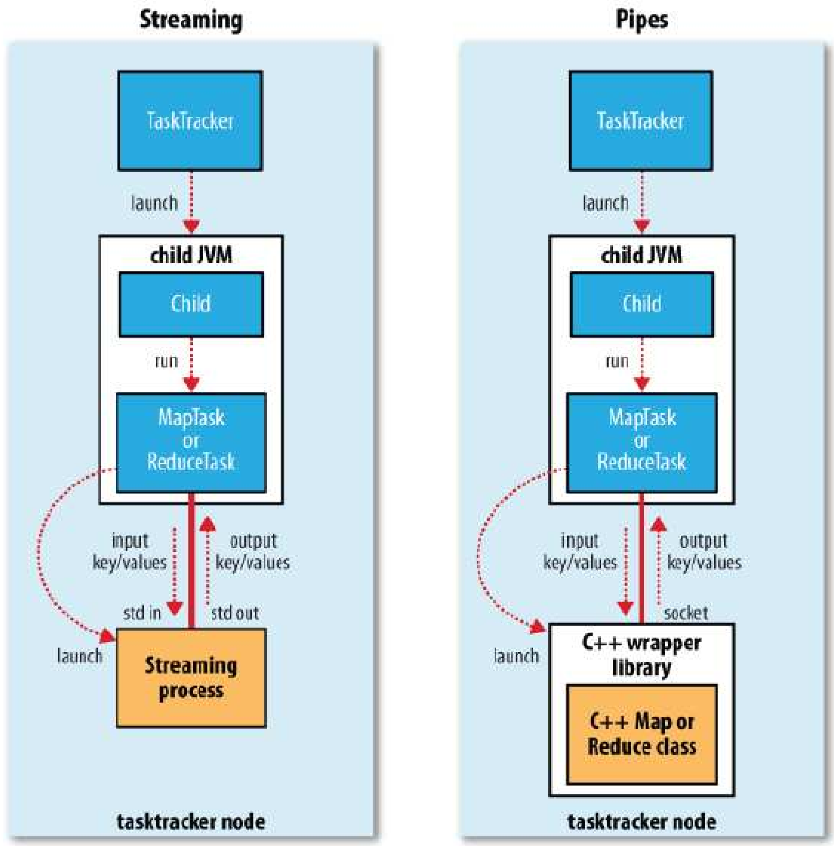


Figura 2.2.4 La relación de los ejecutables Streaming y Pipes en el Tasktracker y su hijo(tomado del Libro Hadoop The Definitive Guide[1]

En el caso de Streaming, la tarea Streaming se comunica con el proceso (que puede ser escrita en cualquier lenguaje) utilizando el método estándar y los flujos de salida. La tarea Pipes, por el contrario, escucha en un socket y pasa el proceso C++ a un número de puerto en su entorno, de modo que en el arranque, el proceso C++ puede establecer una conexión de socket persistentes de nuevo a la tarea padre de Java Pipes.

En ambos casos, durante la ejecución de la tarea, el proceso de Java envía pares clave-valor a los procesos externos, que se ejecutan a través de la función map o reduce definida por el usuario y envía de vuelta a la salida el par clave-valor al proceso de Java. Desde el punto de vista del tasktracker, es como si el proceso hijo tasktracker ejecutó el código en sí del map o reduce.

CAPÍTULO 3

3. DISEÑO E IMPLEMENTACIÓN

3.1. Problemas a resolver

La plataforma Hadoop permite hacer uso de diversos lenguajes de programación tales como: Java, Python y C++, entre otros. Al momento de implementar un programa que realice determinada función sobre la plataforma Hadoop tenemos la libertad de escoger entre estos lenguajes, por lo tanto nuestro estudio, evaluación y comparación permite conocer cuál de los lenguajes previamente mencionados nos brinda un mejor rendimiento en tiempo de ejecución utilizando clúster Hadoop.

Hadoop posee como lenguaje nativo Java, sin embargo ofrece la posibilidad de utilizar otros lenguajes de programación mediante el uso de API's.

Se implementaron 4 programas (WordCounter, Bigramas, Escalado de grises, Hit Log FIEC-ESPOL) en los lenguajes Java, Python y C++ con la finalidad de registrar los tiempos de ejecución de cada uno sobre clústeres de diferentes tamaños.

Los programas WordCounter y Bigramas usaron como dataset de entradas los XML de la Wikipedia con un tamaño de 3GB. Hit Log FIEC-ESPOL utilizó los accesslogs generados por el servidor de correo Ceibo de la FIEC con un tamaño de 3GB y para el escalado de imágenes se utilizaron las fotos de carnet de los estudiantes de la ESPOL con un tamaño de 3GB.

Es importante mencionar que el algoritmo utilizado en las soluciones de los problemas es el mismo para cada lenguaje de programación a fin de obtener mayor precisión en la evaluación de los tiempos de respuesta. A continuación se detalla cada uno de los problemas evaluados, API's y librerías usadas así como el formato de entrada / salida.

3.1.1. WordCounter

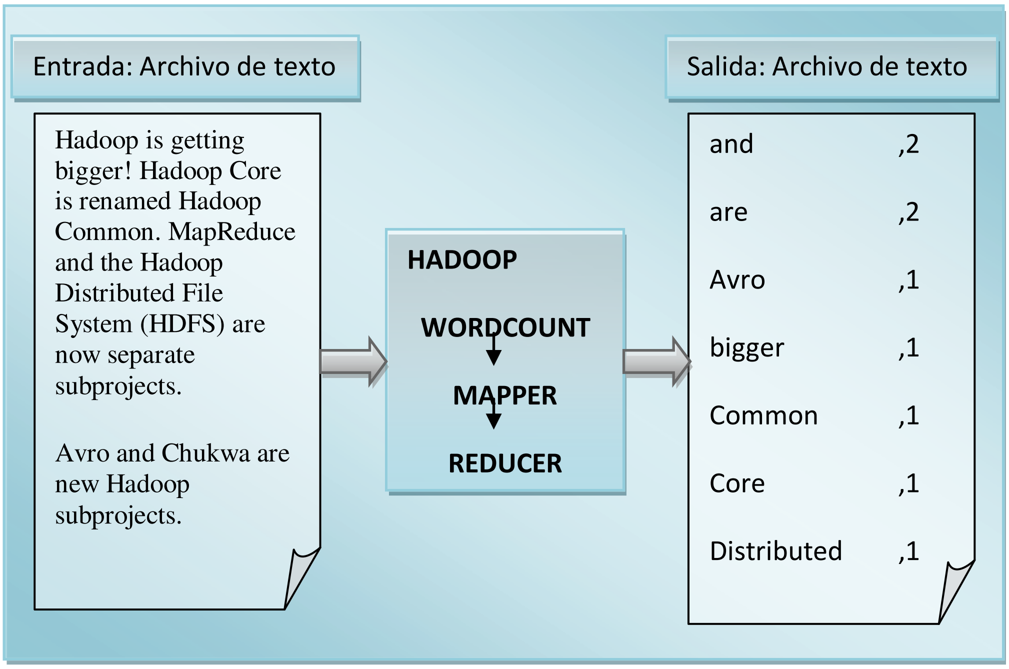
A la hora de escribir un texto, generalmente utilizamos las palabras que más rápido se nos vienen a la mente para explicar algo, por esta razón no es extraño que en ocasiones un término se repita generando redundancias y provocando una lectura con errores gramaticales.

WordCounter es una sencilla aplicación que se encarga de analizar un texto señalando en una lista: las palabras utilizadas y el número de repeticiones existentes.

El formato de los datos entrada/salida es:

Entrada: Archivo texto plano (txt)

Salida : $[\underbrace{\text{ Palabra }}_{\text{Clave}}]\ ,\ [\underbrace{\text{ \# Repeticiones }}_{\text{Valor}}]$



Entrada: Archivo de texto

Hadoop is getting bigger! Hadoop Core is renamed Hadoop Common. MapReduce and the Hadoop Distributed File System (HDFS) are now separate subprojects.

Avro and Chukwa are new Hadoop subprojects.

Salida: Archivo de texto

and	,2
are	,2
Avro	,1
bigger	,1
Common	,1
Core	,1
Distributed	,1

La implementación del WordCountMapper, vía el método mapper, toma cada línea del archivo de texto, según lo dispuesto por el TextInputFormat. La línea es dividida en palabras (tokens) y emite pares clave/valor (palabra, 1).

La implementación del WordCountReducer, vía el método reducer, recibe la salida del mapper que contiene todas las palabras existentes en el archivo de entrada y cuenta la frecuencia de ellas emitiendo un nuevo par clave/valor (palabra,#repetición).

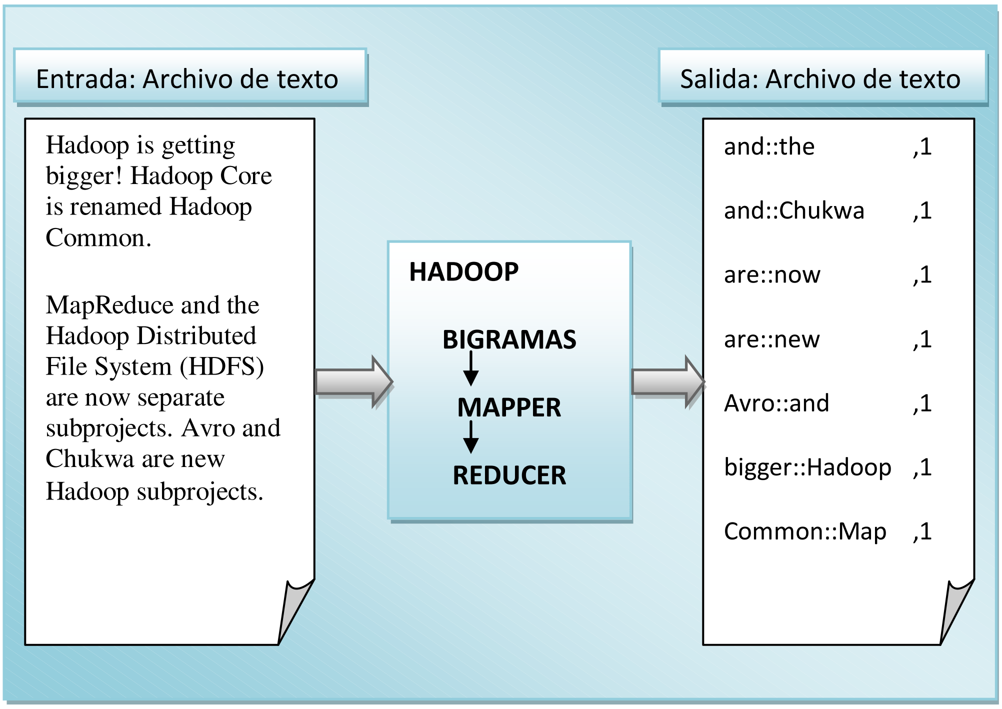
3.1.2. Bigrama

Los Bigramas son grupos de dos letras, dos sílabas, o dos palabras, y son utilizados comúnmente como base para el simple análisis estadístico de texto. Se utilizan en uno de los más exitosos modelos de lenguaje para el reconocimiento de voz.

El formato de los datos entrada/salida es:

Entrada: Archivo texto plano (txt)

Salida : $\underbrace{[\text{ Palabra1, Palabra2 }]}_{\text{Clave}} \;,\; \underbrace{[\text{ \# Repeticiones }]}_{\text{Valor}}$



Hadoop is getting bigger! Hadoop Core is renamed Hadoop Common.

MapReduce and the Hadoop Distributed File System (HDFS) are now separate subprojects. Avro and Chukwa are new Hadoop subprojects.

```
and::the        ,1
and::Chukwa     ,1
are::now        ,1
are::new        ,1
Avro::and       ,1
bigger::Hadoop  ,1
Common::Map     ,1
```

La implementación del BigramaMapper, vía el método mapper, procesa una línea a la vez, según lo dispuesto por el TextInputFormat. De la línea se extrae una palabra y esta es almacenada hasta extraer la siguiente palabra y enviarlas al reducer como pares clave/valor (palabra1:: palabra2, 1).

La implementación del BigramaReducer, vía el método reducer, recibe la salida del mapper que contiene los bigramas existentes en el archivo de entrada y cuenta la frecuencia de ellas emitiendo un nuevo par clave/valor (palabra1 :: palabra2 ,#repetición).

3.1.3. Escalado de grises

El procesamiento de imágenes consiste en alterar una imagen existente de manera determinada. En este caso aplicamos un filtro de escalado de gris a las imágenes provistas como dato de entrada.

Una imagen es un arreglo bidimensional de números, donde cada posición del arreglo representa a un pixel. Las imágenes a color tienen tres bandas o canales correspondiente a los colores R=Red, G=Green, B=Blue. Cada pixel tiene tres valores que representan al RGB.

Uno de los algoritmos para realizar el filtro escalado de grises consiste en promediar los valores RGB de cada pixel.

Parte de la solución aplicada a este problema fue conseguir el *inputFormat* adecuado que permitiera leer todo el archivo (imagen) por completo, no línea a línea como lo hace el formato por defecto usado por Hadoop (*textInputFormat*), para ello usamos el *WholeFileInputFormat* [9] que envía como dato de entrada al mapper todo el contenido del archivo.

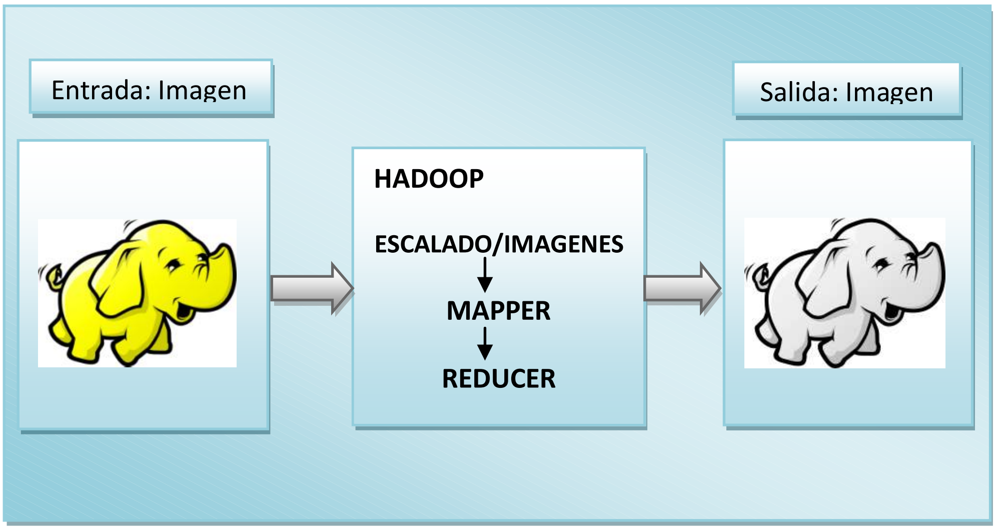
Para aplicar el filtro escalado de gris se hizo uso de varias librerías dependiendo del lenguaje usado:

- Java : JAI (Java Advance Image) [10]
- Python : PIL (Python Image Library) [11]
- C++ : Magick++ (ImageMagick) [12]

El formato de los datos usados es:

Entrada **:** Archivos de imágenes en formatos JPG.

Salida **:** Archivos de imágenes en formatos JPG.



3.1.4. Hit log FIEC-ESPOL

Día a día los sistemas de información generan una información muy valiosa que, por su formato y su volumen, apenas se utiliza, son los logs, que recogen cada una de las aplicaciones y sistemas que configuran la infraestructura de TI.

Los analizadores de logs proporcionan información estadística relevante que puede ser utilizada para toma de decisiones.

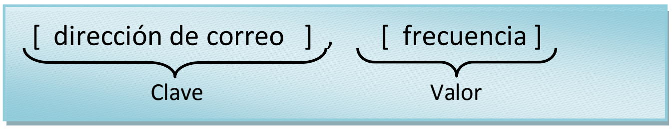
En nuestro caso usamos los logs generados por el servidor de correo Ceibo de la FIEC ESPOL. El análisis que se realizó fue orientado hacia la obtención del listado de cuentas de correo que envían más correos electrónicos.

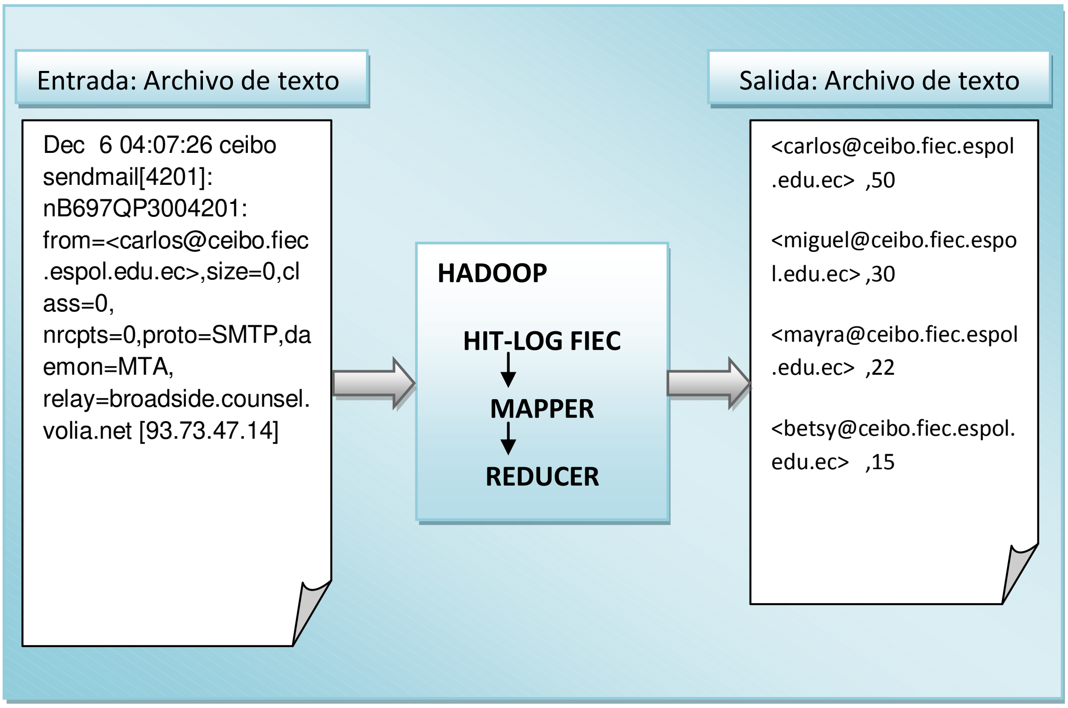
El log utilizado tiene el siguiente formato:

Entrada: Archivo texto plano

```
Dec   6 04:07:26 ceibo sendmail[4201]: nB697QP3004201:
from=<carlos@ceibo.fiec.espol.edu.ec>,size=0,class=0,
nrcpts=0,proto=SMTP,daemon=MTA,
relay=broadside.counsel.volia.net [93.73.47.14]
```

Salida :





3.2. Ejecución

Para realizar la ejecución de las aplicaciones se colocaron todos los dataset necesarios (9GB) a un directorio de la imagen de Fedora iniciada. Además se instalaron las librerías necesarias tales como Image (Python), JAI (Java) y Magick++ (C++) y los compiladores requeridos por Python y C++, cuyo rendimiento fue evaluado en el Anexo A [Tabla de Evaluación de tiempos que toma convertir 1GB de archivos JPG en un solo PC sin usar Hadoop.].

Para ejecutar cada uno de los programas implementados se necesitó una línea de comandos que depende del lenguaje de programación. A continuación se muestra un ejemplo para cada caso.

JAVA:

```
hadoop jar programa.jar input output
```

PYTHON:

```
hadoop jar hadoop-0.20.1-streaming.jar –file mapper.py -mapper mapper.py

-file reducer.py -reducer reducer.py  -input input -output output
```

C++:

```
hadoop pipes -conf word.xml -input input -output output
```

CAPÍTULO 4

4. RESULTADOS Y ANÁLISIS

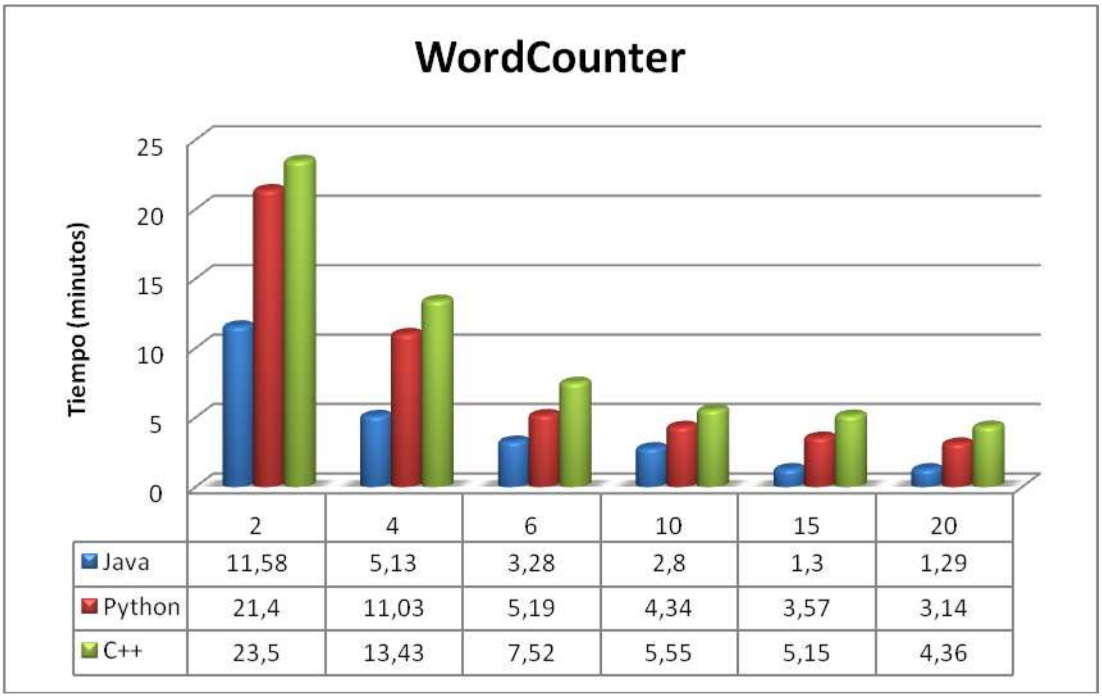
4.1. Resultados

Mediante la interfaz Web proporcionada por Amazon EC2 [16],[17] se tomaron los tiempos de ejecución de cada programa, los cuales se exponen en las siguientes gráficas (Figura 4.1 1 Tiempo de respuestas Vs Número de Nodos aplicación WordCounter, Figura 4.1 2 Tiempo de respuestas Vs Número de Nodos aplicación Bigramas, Figura 4.1 3 Tiempo de respuestas Vs Número de Nodos aplicación Escala de Gris, Figura 4.1 4 Tiempo de respuestas Vs Número de Nodos aplicación Hit Log) que detallan la ejecuciones de las cuatro aplicaciones (WordCounter, Bigrama, Hit Log y Escala de Gris) en los tres lenguajes de programación (Java, Python y C++) en los diferentes tamaños de nodos (2, 4, 6, 10, 15 y 20).

Las gráficas presentan el tiempo en el eje de las ordenadas medido en minutos, mientras que en el eje de las abscisas se muestra el número de nodos en que se ejecutaron las aplicaciones.

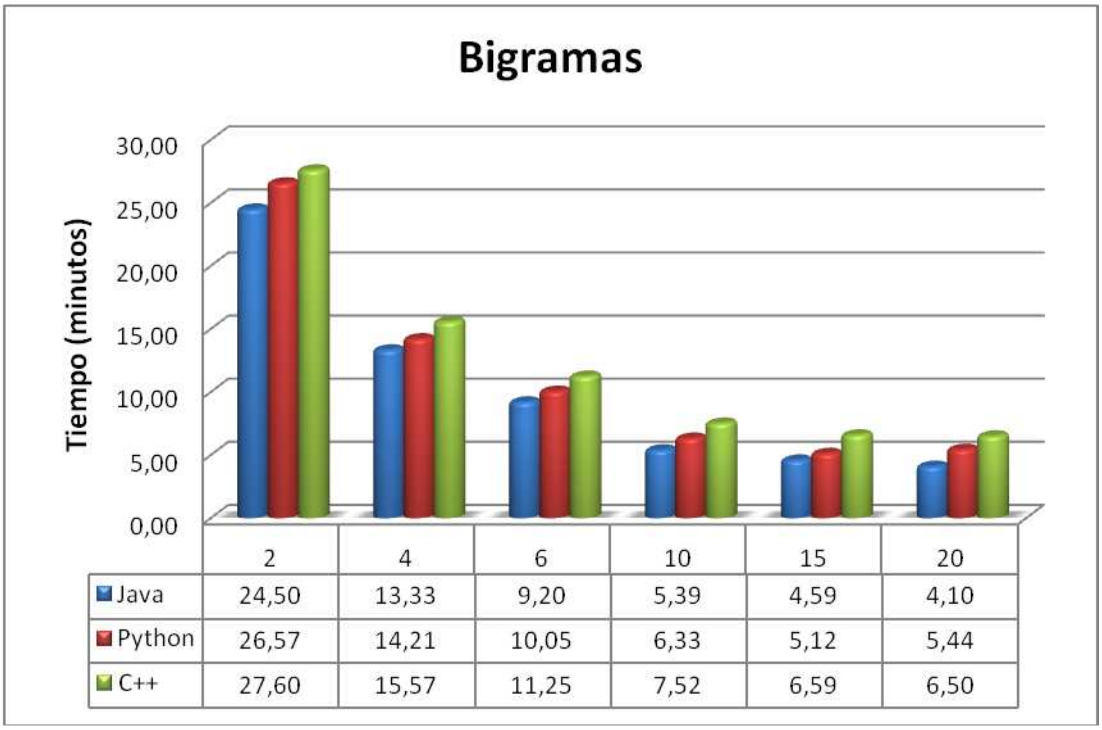
El valor que se grafica en cada barra representa al valor medio obtenido como resultado de la ejecución de cinco veces cada aplicación.

Las gráficas muestran claramente la diferencia entre los tiempos de ejecución registrados en cada lenguaje, que permiten visualizar el lenguaje en determinado programa que obtuvo menor tiempo al ejecutarse.



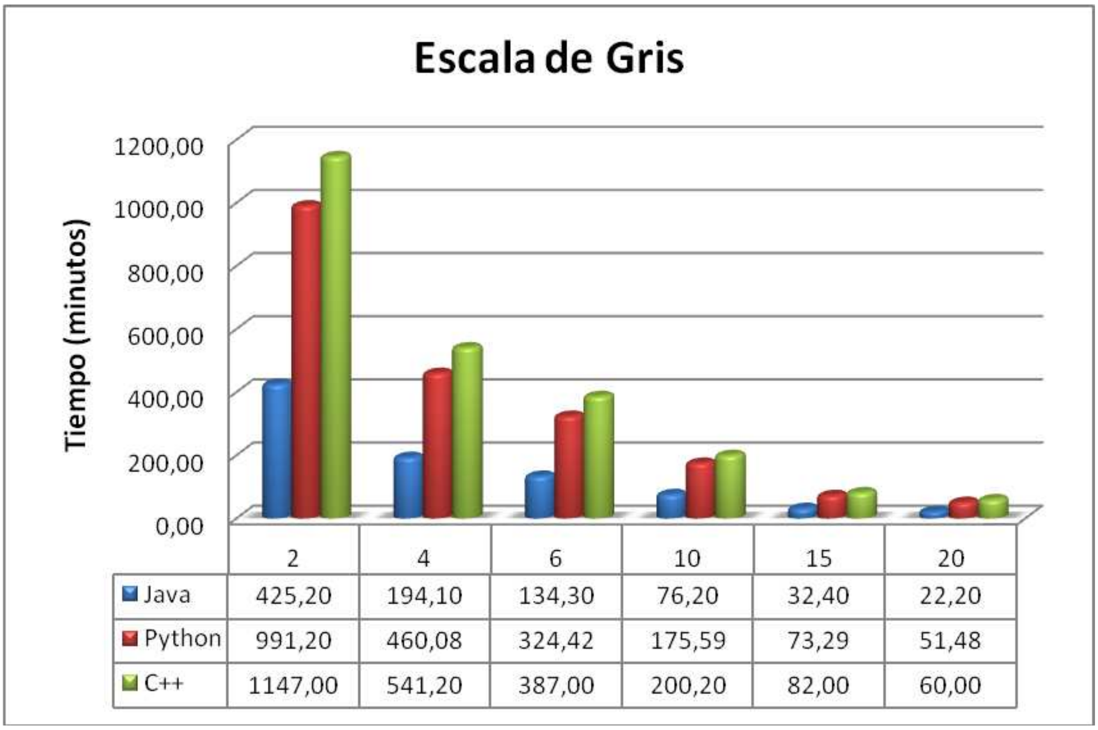
	2	4	6	10	15	20
🟦 Java	11,58	5,13	3,28	2,8	1,3	1,29
🟥 Python	21,4	11,03	5,19	4,34	3,57	3,14
🟩 C++	23,5	13,43	7,52	5,55	5,15	4,36

Figura 4.1 1 Tiempo de respuestas Vs Número de Nodos aplicación WordCounter



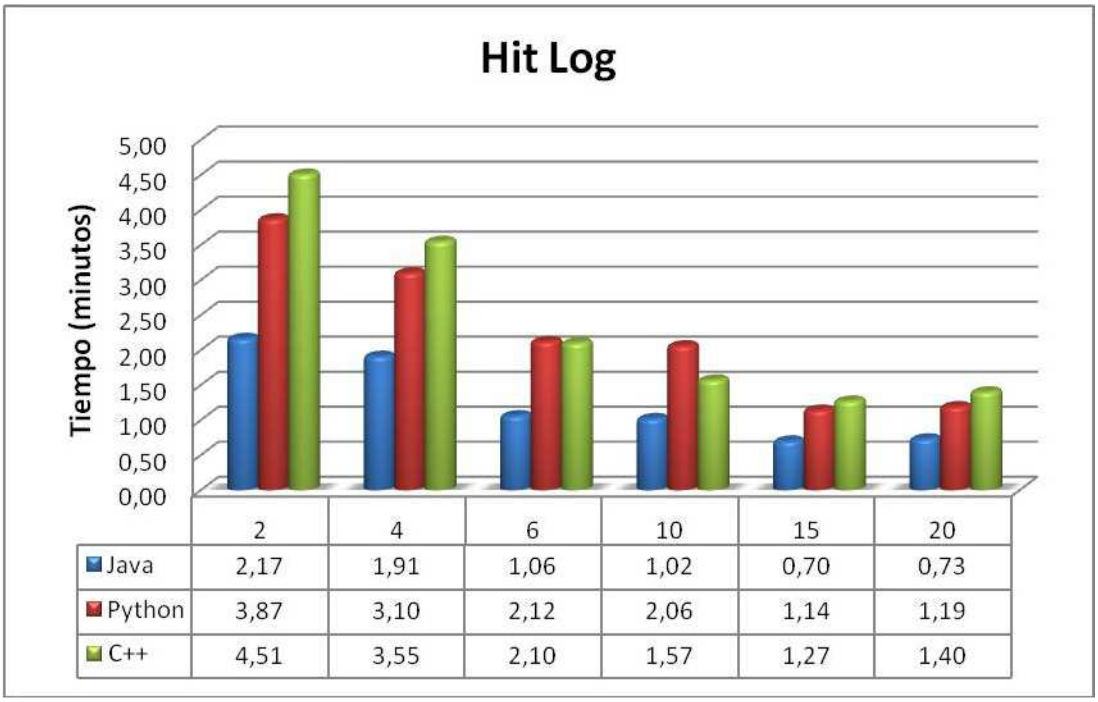
	2	4	6	10	15	20
🟦 Java	24,50	13,33	9,20	5,39	4,59	4,10
🟥 Python	26,57	14,21	10,05	6,33	5,12	5,44
🟩 C++	27,60	15,57	11,25	7,52	6,59	6,50

Figura 4.1 2 Tiempo de respuestas Vs Número de Nodos aplicación Bigramas



	2	4	6	10	15	20
🟦 Java	425,20	194,10	134,30	76,20	32,40	22,20
🟥 Python	991,20	460,08	324,42	175,59	73,29	51,48
🟩 C++	1147,00	541,20	387,00	200,20	82,00	60,00

Figura 4.1 3 Tiempo de respuestas Vs Número de Nodos aplicación Escala de Gris



	2	4	6	10	15	20
🟦 Java	2,17	1,91	1,06	1,02	0,70	0,73
🟥 Python	3,87	3,10	2,12	2,06	1,14	1,19
🟩 C++	4,51	3,55	2,10	1,57	1,27	1,40

Figura 4.1 4 Tiempo de respuestas Vs Número de Nodos aplicación Hit Log

4.2. Análisis

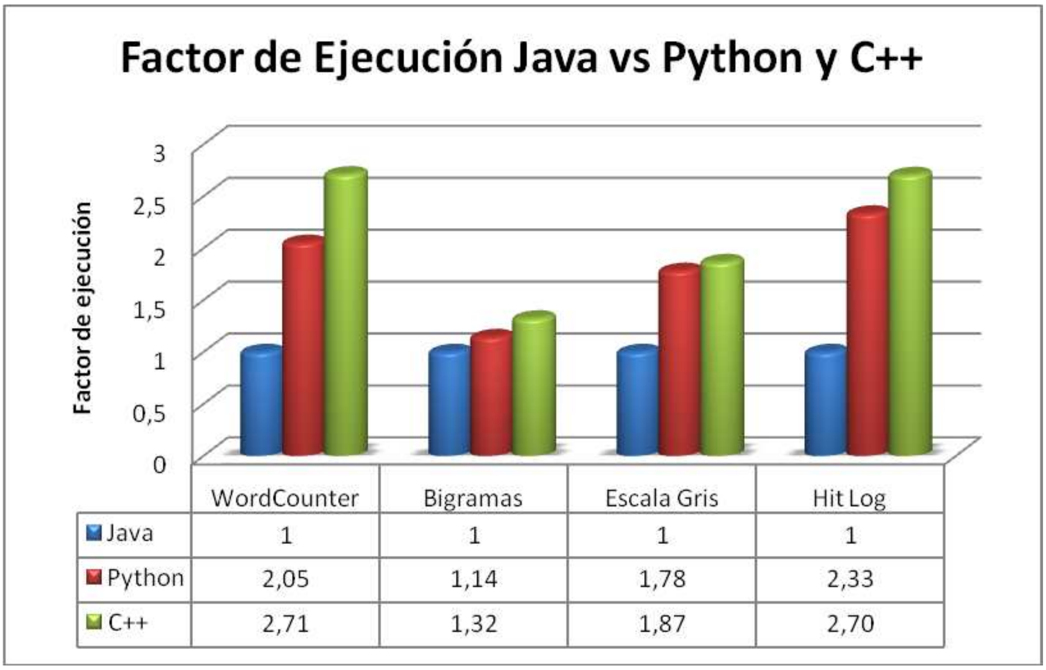
Como se ha detallado en el capitulo teórico del presente trabajo, Hadoop es un framework desarrollado usando lenguaje Java, por lo tanto este es el lenguaje que mayor soporte tiene, sin embargo es posible usar otros lenguajes de programación haciendo uso de API´s o librerías, pero como podemos notar en los resultados obtenidos el tiempo de respuesta haciendo uso de éstos se ve disminuido.

Podemos observar de las gráficas anteriores que Java nativo obtiene el mejor tiempo de respuesta en comparación a Pipes(C++) y

Streaming (evaluado utilizando Python), pero Streaming es más rápido que Pipes. Esta última observación es de interés, ya que se podría pensar que Pipes sería más rápido que Streaming al ser un mecanismo nativo para C++, versus que Streaming es un esquema independiente del lenguaje. Para entender mejor este resultado, conviene analizar nuevamente la Figura 2.2.4 La relación de los ejecutables Streaming y Pipes en el Tasktracker y su hijo. En esta figura podemos ver que la ejecución de un trabajo MapReduce utilizando Pipes involucra un paso adicional a ejecutar el mismo trabajo utilizando Streaming. Ese paso adicional explicaría el menor rendimiento de Pipes observado en los experimentos realizados.

En la Figura 4.2.1 Factor de ejecución Java vs Python y C++ se muestra el factor de ejecución tomando como unidad el tiempo de cada aplicación desarrollada usando Java. Si promediamos dichos factores de tiempo podemos notar que Streaming (evaluado utilizando Python) tarda 1,83 tiempo de Java, mientras que Pipes (C++) tarda 2,15 tiempo Java.

Es decir, Pipes (C++) demora más del doble de tiempo que tarda Java en ejecutar la misma tarea, mientras que Streaming (Python) se presenta en una posición intermedia.



	WordCounter	Bigramas	Escala Gris	Hit Log
🟦 Java	1	1	1	1
🟥 Python	2,05	1,14	1,78	2,33
🟩 C++	2,71	1,32	1,87	2,70

Figura 4.2.1 Factor de ejecución Java vs Python y C++

Cabe mencionar que el tiempo de respuesta de Hit Log (Figura 4.1 4 Tiempo de respuestas Vs Número de Nodos aplicación Hit Log) en comparación a WordCounter (Figura 4.1 1 Tiempo de respuestas Vs Número de Nodos aplicación WordCounter) y Bigrama (Figura 4.1 2 Tiempo de respuestas Vs Número de Nodos aplicación Bigramas) es menor. Esto se debe a que en Hit Log se procesa la línea entera como un solo registro, mientras que para las otras dos aplicaciones la línea es separada en palabras las que se procesan de manera individual.

Otro punto que resaltar en el caso del escalado de gris de las imágenes, es que en promedio cada archivo que se procesó fue de

70KB, es por esto que ésta fue la aplicación que más tiempo tomó en ejecutarse, ya que Hadoop se especializa en procesar archivos de gran tamaño.

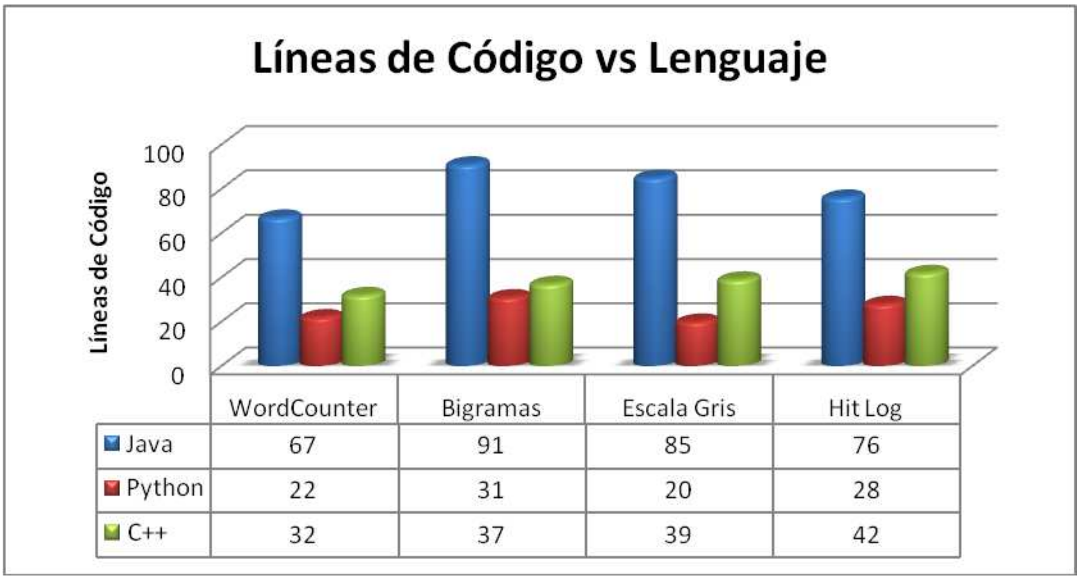
Basándose en los resultados obtenidos parecería que Java es la mejor elección como lenguaje de programación a ser empleado para resolver una tarea específica, sin embargo es importante mencionar que no solo el tiempo de respuesta es un factor a considerar, otro y muy significativo es la experiencia del equipo de desarrollo en dicho lenguaje y el soporte que éste tenga en la plataforma de desarrollo.

Existe poca documentación con respecto a Pipes y Streaming, sin embargo su uso no es muy complicado, pero sí se encuentran limitados con respecto a Java ya que éstos no cuentan con todas las funcionalidades que Hadoop proporcionado para Java. Aunque existen muchos colaboradores que están desarrollando nuevos API´s [18], [19], [20] que permitan a otros lenguajes interactuar de mejor manera con Hadoop, aún los proyectos están en desarrollo y cuentan con poco soporte.

Como se detalla en el Anexo A [Tabla de Evaluación de tiempos que toma convertir 1GB de archivos JPG en un solo PC sin usar Hadoop.], el tiempo de ejecución de la librería de C++ es mayor, esto denota otra razón por la que Pipes(C++) registró los tiempos más altos en la evaluación.

De nuestra experiencia podemos compartir que a pesar de no haber usado Python antes al presente trabajo, el aprendizaje de éste lenguaje fue bastante rápido y sencillo. Además las líneas de código utilizadas para resolver el mismo problema en comparación de Java o C++ fueron reducidas.

En la Figura 4.2 2 Número de líneas de Código vs Lenguaje de programación se puede observar el resumen gráfico del número de líneas de código utilizadas para resolver cada una de las aplicaciones usando el mismo algoritmo de solución.



	WordCounter	Bigramas	Escala Gris	Hit Log
🟦 Java	67	91	85	76
🟥 Python	22	31	20	28
🟩 C++	32	37	39	42

Figura 4.2 2 Número de líneas de Código vs Lenguaje de programación

La sintaxis de cada uno de los lenguajes que están siendo objeto de estudio en esta tesis implicó una variación significativa en el número de líneas de código (18% Python, 26% C++ y 56% Java). Ya que algunos lenguajes están diseñados para realizar operaciones de la manera más sencilla posible mientras que otros no.

CONCLUSIONES Y RECOMENDACIONES

Conclusiones

1. En las aplicaciones sujetas a análisis en este trabajo podemos notar que el API de programación de Hadoop de mejor rendimiento en tiempo de respuesta fue Java nativo seguido por Streaming (evaluado con Python) y finalmente Pipes (C++).

2. A más de considerar el tiempo de respuesta de un lenguaje de programación para efectuar una elección, debemos también considerar también las preferencias/conocimientos del desarrollador y el soporte disponible a fin de equilibrar rendimiento vs facilidad de desarrollo.

3. El procesamiento de texto en Hadoop fue realizado con el uso de manejo de cadenas que básicamente consistió en separar el texto y analizarlo para cada lenguaje existen diferentes formas de hacerlo.

4. El procesamiento de archivos pequeños (en el orden de los KB) demora la tarea de ejecución debido a que Hadoop está desarrollado para ser más eficiente manipulando archivos de gran tamaño.

5. Elegir el lenguaje de programación a usar dependerá de la complejidad de la aplicación. Si necesitamos una aplicación compleja, que requiera acceso al HDFS por ejemplo elegiríamos Java, debido a que posee soporte para este tipo de operaciones. Pero, si se desea hacer procesamiento simple de texto definitivamente por su sencillez elegiríamos Streaming con un lenguaje como Python.

Recomendaciones

1. Hadoop está desarrollado de tal manera que es mucho más eficiente trabajando con archivos de gran tamaño (en el orden de los GB, TB) es por ello que existe el SequenceFile [13], que permite unir muchos archivos pequeños (KB) en un solo archivo grande (GB, TB). Es recomendable hacer uso de este InputFormat cuando los requerimientos de la aplicación a desarrollar lo permitan.

2. Si se trabajan con InputFormat propio, es recomendable empaquetarlos junto a los demás InputFormat provistos por Hadoop de manera que no existan inconvenientes al momento de ser referenciado.

3. Al momento de realizar concatenación de cadenas de palabras es mejor utilizar los métodos que brinda cada lenguaje para hacerlo, ya que algunos lenguajes permiten el uso del símbolo “+” que realiza la misma acción pero toma más ciclos de reloj del sistema operativo para hacerlo.

REFERENCIAS BIBLIOGRÁFICAS

[1]. Andrew Pavlo, Erick Paulson, Alexander Rasin, Daniel J. Abadi, David J. DeWitt, Samuel Madden, Michael Stonebraker, “A Comparison of Approaches to Large-Scale Data Analysis”, Brown University, University of Wisconsin, 2009.

[2]. Xiaoyang Yu, “Estimating Language Models Using Hadoop and Hbase”, University of Edinburgh, 2008.

[3]. The Apache Software Foundation, Apache Lucene, http://lucene.apache.org/, último acceso 17-Feb-2010.

[4]. Dean, J. y Ghemawat, S. “MapReduce: Simplified Data Processing on Large Clusters”. En memorias del Sixth Symposium on Operating System Design and Implementation (OSDI 2004), San Francisco, CA-EE.UU. Diciembre 2004.

[5]. Dhruba Borthakur, HDFS Architecture, http://hadoop.apache.org/common/docs/current/hdfs_design.html, último acceso 17-Feb-2010.

[6]. Ghemawat, S., Gobioff, H., y Leung, S. “The Google File System”. En Memorias del 19th ACM Symposium on Operating Systems Principles. Lake George, NY-EE.UU., Octubre, 2003.

[7]. The Apache Software Foundation, Welcome to Apache Hadoop, http://hadoop.apache.org/core/ , último acceso 17-Feb-2010.

[8]. Tom White, Anatomy of a MapReduce job run with Hadoop, http://answers.oreilly.com/topic/459-anatomy-of-a-mapreduce-job-run-with-hadoop/, último acceso 20-Abr-2010.

[9]. Hadoop The definitive Guide, Tom White, publicado por O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472. Pag. 192, Junio 2009.

[10]. Oracle Corporation, Java Advance Imaging (JAI) API, http://java.sun.com/javase/technologies/desktop/media/jai/, último acceso 17-Feb-2010.

[11]. PythonWare, Python Imaging Library (PIL), http://www.pythonware.com/products/pil/, último acceso 17-Feb-2010.

[12]. ImageMagick Studio LLC, Magick++ -- C++ API for ImageMagick, http://www.imagemagick.org/Magick++/, último acceso 17-Feb-2010.

[13]. The Apache Software Foundation, Sequencefile, http://hadoop.apache.org/common/docs/current/api/org/apache/hadoop/io/SequenceFile.html, último acceso 5-Mar-2010.

[14]. Hadoop The definitive Guide, Tom White, publicado por O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472. Pag. 184, Junio 2009.

[15]. Hadoop The definitive Guide, Tom White, publicado por O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472. Pag. 20, Junio 2009.

[16]. Michael G. Noll, MapReduce job tracker(s), http://IP de maquina virtual:50030, último acceso 5-Mar-2010.

[17]. Michael G. Noll, HDFS name node(s), http://IP de maquina virtual:50070, último acceso 5-Mar-2010.

[18]. Audioscrobbler Ltd, Dumbo, http://www.audioscrobbler.net/development/dumbo/, último acceso 5-Mar-2010.

[19]. Google Project Hosting, Python API to HDFS: libpyhdfs, http://code.google.com/p/libpyhdfs/, último acceso 5-Mar-2010.

[20]. The Apache Software Foundation, C API to HDFS:libhdfs, http://hadoop.apache.org/common/docs/r0.20.1/libhdfs.html, último acceso 5-Mar-2010.

[21]. Hadoop The definitive Guide, Tom White, publicado por O’Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472. Pag. 154, Junio 2009.

[22]. Hadoop The definitive Guide, Tom White, publicado por O’Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472. Pag. 157, Junio 2009.

ANEXO A

1. Tabla de Evaluación de tiempos que toma convertir 1GB de archivos JPG en un solo PC sin usar Hadoop.

	Tiempo (min)
Python	38,57
Java	62,58
C++	125

Podemos observar que Python utilizando PIL tuvo el mejor rendimiento, seguido de Java utilizando JAI.