ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

Facultad de Ingeniería en Electricidad y Computación"Utilización de la plataforma Hadoop para la detección de potencial plagio con indicadores de probabilidad de certeza de las tareas enviadas a un Sistema de Administración de Cursos (aplicable para SIDWeb o Metis)"

INFORME DE MATERIA DE GRADUACIÓN

Previa a la obtención del Título de:

INGENIERO EN CIENCIAS COMPUTACIONALES

ESPECIALIZACION SISTEMAS INFORMACIÓN

Presentada por:

EDUARDO SEGUNDO CRUZ RAMÍREZ

DIEGO ARMANDO LAVAYEN ALARCÓN

Guayaquil - Ecuador

2010

AGRADECIMIENTO

- Principalmente a Dios por permitirnos llegar hasta donde hemos llegado y por impulsarnos a alcanzar nuevas metas.
- A nuestros padres, los mejores regalos que Dios nos dió, pilares fundamentales en nuestra formación personal y profesional.
- A la MSc. Cristina Abad Robalino, por sus constantes consejos y ayuda a lo largo del proyecto.
- A la MBA. Ana Tapia Rosero, por su gentil ayuda en la culminación del presente.
- Al MSc. Federico Raue, por sus muy útiles recomendaciones que mejoraron el performance de nuestra propuesta.
- A nuestros amigos que nos brindaron su apoyo en momentos cruciales dentro del proyecto.

DEDICATORIA

Dedicamos enteramente

este trabajo a nuestros padres,

personas de lucha incansable que

nos enseñaron que con perseverancia y

paciencia se puede llegar a donde uno desee…

TRIBUNAL DE SUSTENTACIÓN

MBA. Ana Tapia Rosero

PROFESORA DE LA MATERIA DE GRADUACIÓN

MSc. Federico Raue R.

PROFESOR DELEGADO POR EL DECANO DE LA FACULTAD

DECLARACIÓN EXPRESA

"La responsabilidad del contenido de este Proyecto de Graduación, nos corresponde exclusivamente; y el patrimonio intelectual de la misma, a la Escuela Superior Politécnica del Litoral"

(Reglamento de exámenes y títulos profesionales de la ESPOL)

Eduardo Segundo Cruz Ramírez

Diego Armando Lavayeń Alarcón

RESUMEN

En el presente trabajo se expone un informe del análisis, diseño, implementación y pruebas del módulo para la detección de potencial plagio de las tareas enviadas a un Sistema de Administración de Cursos, utilizando como base tecnológica la programación en paralelo sobre la plataforma de Hadoop, que podría ser adaptado al SIDWeb o Metis.

El documento se encuentra dividido principalmente en cinco capítulos que, en su totalidad, exponen los aspectos teóricos y técnicos utilizados para comprender el porqué y el cómo se desarrolló este tema.

En el primer capítulo, se define la problemática que se desea resolver indicando el objetivo general y sus respectivos objetivos específicos que planteamos al inicio de esta propuesta. Este capítulo determina una meta concreta y esboza las directrices procedimentales que guiarán el proyecto, acotadas por las limitantes intrínsecas y extrínsecas del desarrollo del mismo.

En el segundo capítulo, se presenta un análisis de la base conceptual que utilizamos para comprender cómo la necesidad de comparar dos cadenas está presente en otras ramas de la ciencia, como en la biología, y cómo la solución ha sido propuesta con el uso de herramientas informáticas; así mismo, se expone la estrategia para realizar alineamientos locales de secuencias

biológicas con el uso del algoritmo de Smith-Waterman[1] y cómo éste resulta de interés en nuestro trabajo como base de la propuesta del PhD. Robert W. Irving[2] en el que se realiza una mejora para maximizar la cantidad de alineamientos resultantes a partir de dos cadenas sujetas a comparación.

En el tercer capítulo, se expone conceptualmente las tecnologías utilizadas para llevar a cabo el proyecto, tanto como el servicio de almacenamiento escalable de datos ofrecido por Amazon (S3), la infraestructura con capacidad de cómputo variable (también de Amazon) para el procesamiento de aplicaciones flexibles tolerante a fallos (EC2), la plataforma utilizada para el procesamiento masivo de datos (Hadoop) y el modelo de programación Map/Reduce, que proponemos para el desarrollo de este proyecto.

El cuarto capítulo detalla específicamente como se hizo frente a la problemática expuesta en la implementación del módulo, resultado de la unión conceptual de los puntos citados en los capítulos dos y tres, dividiendo el proceso en dos partes básicamente:

- ***Primero:*** el pre-procesamiento de los archivos del Sistema de Administración de Cursos para generar archivos en texto plano similares a sus fuentes en los que se conservaron sólo las palabras no consideradas como vacías o carentes de significado semántico y con sólo

caracteres trascendentes (*caracteres en el rango de la **'a'** a la **'z'**, de la **'A'** a la **'Z'**, del **'0'** al **'9'***).

- ***Segundo:*** la implementación del algoritmo de Smith-Waterman con las mejoras planteadas por PhD. Robert W. Irving para determinar el plagio haciendo uso de la plataforma de Hadoop con su modelo de programación Map/Reduce.

En el quinto capítulo se expone un resumen de las pruebas realizadas y el análisis comparativo obtenido a partir de éstas, permitiendo establecer empíricamente cómo, con el uso de más nodos y una cantidad constante de datos, es posible reducir el tiempo promedio de cómputo total.

Al finalizar el presente trabajo proponemos nuestras conclusiones y las recomendaciones para futuras implementaciones, a partir de los problemas presentes y los resultados obtenidos en la elaboración del mismo.

ÍNDICE GENERAL

ÍNDICE DE GRÁFICOS

GLOSARIO

ADN: Ácido Desoxirribonucleico, es un tipo de ácido nucleico, que forma parte de todas las células. Contiene la información genética usada en el desarrollo y el funcionamiento de los organismos vivos conocidos y de algunos virus, siendo el responsable de su transmisión hereditaria. Desde el punto de vista químico, es un polímero de nucleótidos, es decir, un polinucleótido. Un polímero es un compuesto formado por muchas unidades simples (nucleótidos) conectadas entre sí formados por un azúcar (*la desoxirribosa*), una base nitrogenada (*que puede ser adenina→A, timina→T, citosina→C o guanina→G*) y un grupo fosfato que actúa como enganche de cada unidad con el siguiente.[1]

ARN: Ácido ribonucleico, es un ácido nucleico formado por una cadena de monómeros repetitivos llamados nucleótidos, formados de un monosacarido, una base nitrogenada, y un grupo fosfato. [2]

Clúster: Este término se aplica a los conjuntos de computadoras construidos mediante la utilización de componentes de hardware comunes y que se comportan como si fuesen una única computadora cuya tecnología ha evolucionado en apoyo de actividades que van desde aplicaciones de

[1] http://es.wikipedia.org/wiki/ADN
[2] http://es.wikipedia.org/wiki/ARN

supercómputo y software de misiones críticas, servidores web y comercio electrónico, hasta bases de datos de alto rendimiento, entre otros usos.

El cómputo con clústeres surge como resultado de la convergencia de varias tendencias actuales que incluyen la disponibilidad de microprocesadores económicos de alto rendimiento y redes de alta velocidad, el desarrollo de herramientas de software para cómputo distribuido de alto rendimiento, así como la creciente necesidad de potencia computacional para aplicaciones que la requieran.[3]

DNS: Sistema de nombre de dominio, es un sistema de nomenclatura jerárquica para computadoras, servicios o cualquier recurso conectado al internet o a una red privada. Este sistema asocia información variada con nombres de dominio asignado a cada uno de los participantes. Su función más importante, es resolver nombres inteligibles para los humanos en identificadores binarios asociados con los equipos conectados a la red, esto con el propósito de poder localizar y direccionar estos equipos mundialmente. [4]

Procesamiento distribuido: una colección de computadoras separados físicamente y conectados entre sí por una red de comunicaciones distribuida; cada máquina posee sus componentes de hardware y software que el usuario

[3] http://es.wikipedia.org/wiki/Cluster_(informática)
[4] http://es.wikipedia.org/wiki/DNS

percibe como un solo sistema (no necesita saber qué cosas están en qué máquinas). El usuario accede a los recursos remotos (RPC) de la misma manera en que accede a recursos locales, o un grupo de computadores que usan un software para conseguir un objetivo en común.

Los sistemas distribuidos deben ser muy confiables, ya que si un componente del sistema se descompone otro componente debe de ser capaz de reemplazarlo, esto se denomina Tolerancia a Fallos.[5]

Palabras vacías: o "*stopwords*" (como son conocidas en inglés) son un conjunto de palabras carentes de sentido semántico en el procesamiento de datos en lenguaje natural.

Aunque no se ha definido una lista de palabras vacías que todas las herramientas de procesamiento de lenguajes natural incorporen, se puede generalizar su identificación a los: artículos, pronombres, preposiciones, etc.. [6]

[5] http://es.wikipedia.org/wiki/Computación_distribuida
[6] http://es.wikipedia.org/wiki/Palabras_vacías

INTRODUCCIÓN

Las principales problemáticas presentes en colegios y universidades en la actualidad son:

a) El alto índice de incidencia de copias en la presentación de tareas,
b) La no compleja pero si laboriosa tarea de discriminar trabajos que hayan sido sujetos a copia.
c) Determinar cuán alto ha sido el nivel de copia en una tarea para definir una calificación justa (*sujeto obviamente a juicio del profesor*).

La gran incidencia de copias en la presentación de tareas, principalmente a nivel universitario, se debe al uso indiscriminado de fuentes de información que están en Internet como la Wikipedia[7], sin que el trabajo esté sujeto a una intervención analítica del mismo y se reduzca simplemente a una copia directa del sitio.

Este tipo de copias no sólo representan una infracción a los derechos de autor de cualquier artículo utilizado (*penalizado de por sí en muchos países*), sino también una falta de madurez y honradez por parte del estudiante cuya conducta debería ser perfilada y corregida por el docente.

[7] http://es.wikipedia.org/wiki/Wikipedia

Por otro lado, la labor de monitorear si una tarea ha sido sujeta a copia no es del todo compleja pues se resume a la simple comparación sintáctica de un deber contra el resto de trabajos enviados en un paralelo en particular pero esto puede demandar una cantidad estimable de tiempo dependiendo de la extensión de la tarea y la cantidad de respuestas a la misma.

Ahora, si se consideran los obstáculos que pueden aparecer en la presentación de las tareas como diferencia en extensiones, formatos, imágenes, carátulas y demás artilugios que los estudiantes suelen añadir a sus trabajos para imprimirle su “marca personal”, la simple comparación sintáctica puede convertirse en una tediosa labor que demanda demasiado tiempo.

Estos aspectos son los que, entre otros, nos empujaron a plantear y desarrollar un módulo que permita agilitar el proceso de detección de plagio y provea indicadores de potenciales copias determinando así, no sólo que trabajo fue plagio de otro (*con datos que puedan sugerir este orden*), sino también en qué porcentaje fue realizada dicha copia, para que así el docente pueda tomar decisiones de corrección y calificación basadas en datos concretos y extraídos de las mismas tareas de sus estudiantes.

CAPÍTULO 1

1. Planteamiento del Problema

1.1. Definición del problema

En la actualidad, con el auge de recursos informáticos de libre acceso en el Internet y con el creciente número de tareas en formato digital que se solicitan en las universidades es común encontrar copias totales o parciales que pueden pasar desapercibidos por los profesores, no porque éstos no analicen los trabajos que sus estudiantes presentan, sino por la "*astucia*" que muestran los estudiantes al presentarlos.

Entre las técnicas que son utilizadas para esconder una copia tenemos:

- Cambios de formato en el texto,
- Cambios de extensiones en los documentos,
- Inserciones de más contenido (*procedente tal vez de otra copia*),

- Intercambio en el orden de los párrafos sin alterar el contenido semántico de la tarea,
- Atavío de copias con imágenes, marcos, colores, entre otros elementos que le imprimen una marca única al deber pero que no lo aleja de su realidad.

Ahora, que sucedería si la copia no se realiza de una tarea circunscrita en un paralelo en particular, sino que se utilizan fuentes (*que no necesariamente son tareas, sino recursos enviados por los profesores*) de otros paralelos, cursos o incluso años bajo la premisa que: "*yo soy de otro paralelo y no se darán cuenta si te copio*" o "*yo ya pasé esa materia, ya te presto todas las tareas que me enviaron*".

No porque resulte difícil identificar este tipo de copia significa que la acción deja de ser deshonesta y el involucrado en este acto debe quedar sin reprimenda.

Una solución sería revisar manualmente el contenido de cada tarea comparándola con todos los recursos existentes en el Sistema de Administración de Cursos, pero esta labor es virtualmente imposible.

Hasta ahora no se puede determinar con eficiencia y eficacia si este tipo de copias son cometidas, permitiendo de manera inconscientemente esta actividad.

1.2. Objetivo general

Desarrollar un módulo de detección de potencial plagio, con indicadores de probabilidad de certeza, de las tareas enviadas a un Sistema de Administración de Cursos (aplicable a SIDWeb o Metis) utilizando como recurso informático el paradigma de programación Map/Reduce sobre la plataforma de Hadoop.

1.3. Objetivos específicos

- Implementar un proceso que convierta documentos de extensiones: .doc, .docx, .ppt, .pptx, .xls, .xlsx, .pdf, .html, .htm, .txt; en archivos de texto plano, en los que se eliminen las palabras vacías y caracteres no relevantes.
- Almacenar cada archivo de texto plano con un código único que lo identifica con su original.
- Implementar el algoritmo de Smith-Waterman, bajo el paradigma de programación Map/Reduce, para la detección de coincidencias significativas entre el contenido de dos archivos de texto plano.
- Determinar en qué grado un archivo de texto plano es copia de otro u otros archivos.

1.4. Justificación

Un sistema que monitoree si existió plagio entre las tareas enviadas en un paralelo en particular, no demandaría la utilización de procesamiento masivo de datos como se expone en este proyecto, pero el objetivo de este módulo es un poco más ambicioso ya que se propone revisar si un conjunto de tareas han sido sujetas a copias, no sólo entre ellas, sino entre todas las enviadas en cualquier paralelo de cualquier curso de cualquier año.

Es por esta principal razón por la que se decidió extender la implementación del presente, en un escenario distribuido utilizando la plataforma de Hadoop, puesto que no tendremos volumen de datos por peso individual de archivo, sino por cantidad total de data a procesarse (86000 archivos de texto plano).

Implementando el algoritmo de Smith-Waterman con las mejoras de PhD. Robert W. Irving y ejecutándolo de manera local comparando dos archivos de 16.56 Kb y 16.35 Kb respectivamente, se registró un tiempo medio de 3.27 segundos.

Si escalamos éste valor a la comparación secuencial de 86000 archivos, tendríamos un tiempo medio de 281220 segundos (*3 días 6 horas y 7 minutos*), tiempo que notoriamente está por encima de lo aceptable y demanda la búsqueda de alternativas de procesamiento distribuido que agilite de manera eficaz el proceso.

El desarrollo de este módulo, que podría ser adaptado a Sistemas de Administración de Cursos implementados en la Institución (SidWEB o Metis), pondría a disposición de los profesores una herramienta que les permita identificar con mayor facilidad, eficiencia y eficacia qué trabajos tienen altos niveles de probabilidad de copia para que así puedan tomar las medidas correctivas del caso.

1.5. Alcances y limitaciones

Se desarrolló un sub-módulo que pre-procesará los archivos que utilizamos en las pruebas de detección de plagio. Este sub-módulo no fue desarrollado con técnicas de procesamiento masivo porque en un ambiente de producción la conversión de éstos a archivos de texto plano no demandaría muchos recursos o tiempo de procesamiento en el servidor del Sistema de Administración de Cursos; puede ser adaptado como una funcionalidad adicional que realice este trabajo en segundo plano inmediatamente después de que una tarea haya sido subida al Sistema, para luego obtener y almacenar el resultado en una ruta particular en S3 bajo un nombre único que identifique el archivo y que lo relacione con su original.

El módulo propuesto, determina un valor en porcentaje que indica la probabilidad de certeza que tienen dos documentos en ser iguales, basándose

en la implementación del algoritmo de Smith-Waterman con las modificaciones del PhD. Robert W. Irving.

Una limitación que se encontró para las pruebas de este módulo, es la falta de acceso a una base de datos que administre los cursos, paralelos y las tareas, en la que nos indique la relación que existe entre estos y las fechas y horas de envío para poder simular con mayor facilidad un entorno de producción normal y así determinar con exactitud, basándose en horas de envío, que la tarea **A** es copia de **B** y no al contrario.

CAPÍTULO 2

2. Análisis Conceptual

2.1. Alineamiento de secuencias

Una secuencia es un conjunto de elementos encadenados uno detrás de otro; en el área de bioinformática las cadenas de **ADN**, **ARN** o estructuras primarias proteicas son representadas como una secuencia de caracteres donde cada caracter representa un aminoácido o nucleótido.

El alineamiento de secuencias es utilizado en bioinformática como un proceso en el que se representa y compara dos o más cadenas proteicas en busca de segmentos donde exista coincidencia entre ellos.

A continuación vamos a ilustrar el proceso de alineamiento de secuencias, para ellos vamos a tomar como ejemplo dos secuencias que no se encuentran alineadas:

Figura 1 Secuencia sin alinear

Tomando como base la Figura 1 podemos apreciar que sin necesidad de aplicar el proceso de alineamiento ya existen coincidencias en las mismas posiciones de los caracteres. Cuando existe la coincidencia entre un carácter de ambas secuencias en la misma posición es denominado como un *"hit"*.

En la Figura 2 podemos apreciar que en la cadena sin alinear existen 3 hits.

Figura 2 Hits en una secuencia sin alinear

El proceso de alinear secuencias consiste en agregar espacios en blancos a las secuencias de tal manera que desplace a los caracteres siguientes. El objetivo de este desplazamiento es lograr coincidencias en otras posiciones de la secuencia. El término con el cual se define a la inserción de un espacio en blanco es el *"indel"* (*llamado así por su correspondiente en inglés* ***in****sert* ***del****et*).

Tomando como ejemplo las cadenas iniciales sin alinear en la Figura 1. Si insertamos un espacio al inicio de la primera cadena logramos el desplazamiento de los demás caracteres Figura 3 y a su vez obtenemos 6 *"hits"* como podemos ver en la Figura 4.

Figura 3 Agregando un indel en la primera secuencia

Figura 4 Hits generados a partir del indel en la primera secuencia

En cambio si agregamos *"indels"* a ambas cadenas, como podemos apreciar en la Figura 5, logramos que la cantidad de alineamientos sea mayor obteniendo así 9 *"hits"* como se muestra en la Figura 6.

Figura 5 Agregando indels a ambas secuencias

Figura 6 Hits generados a partir de los indels en ambas secuencias

Cuando no existe una coincidencia en la misma posición de ambas cadenas se lo denomina como un *"replace"* como se muestra en la Figura 7.

Figura 7 Replaces de las secuencias alineadas

El proceso de alinear dos cadenas está sujeto a encontrar cual es el alineamiento mas óptimo de entre todos los posibles alineamientos, para ello es necesario definir un **Sistema de Puntuación** en el que se define una ponderación para cada uno de los términos antes mencionados (*"hit"*, *"indel"*,

“replace”). Esta ponderación es multiplicada por cada una de las cantidades resultantes de cada uno de los factores y al final se suma todos estos valores de tal manera que nos da como resultado una puntuación.

Una vez definido el sistema de puntuación, un alineamiento óptimo es aquel que brinde un mayor resultado.

De acuerdo al estudio del PhD. Robert Irving[2] para la detección de plagio los mejores valores a tomar en el sistema de puntuación son *“hit”* = 1, *“indel”* =-1, *“replace”* = -1, de esta manera se castiga los *“indels”* y los *“replaces”* y se premia a los *“hits”*.

A continuación se muestra el cálculo del sistema de puntuación de las secuencias anteriormente alineadas.

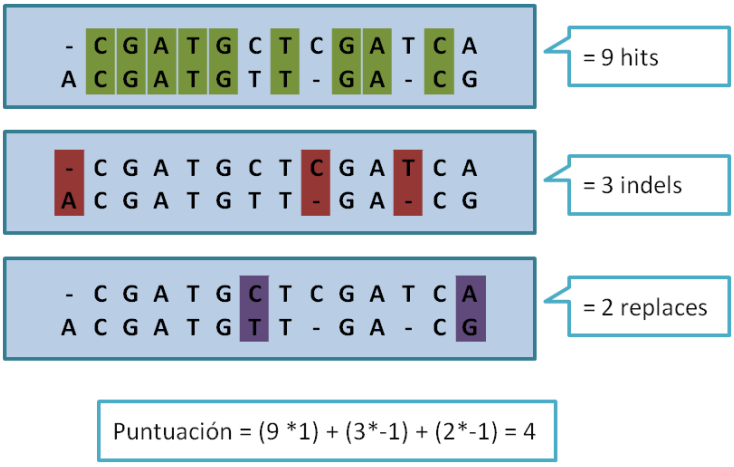


Figura 8 Sistema de puntuación aplicado a un alineamiento

2.1.1. Alineamientos Globales

El alineamiento global es aquel que se extiende a lo largo de toda la longitud de la secuencia. Este alineamiento es muy utilizado cuando las secuencias son similares y el tamaño es aproximadamente el mismo. El proceso consiste en agregar espacios a las secuencias logrando la mayor cantidad de coincidencias a lo largo de ambas secuencias.

Un algoritmo que utiliza alineamiento global es el planteado por Needleman-Wunsch[8] basado en programación dinámica.

2.1.2. Alineamientos Locales

Los alineamientos locales son utilizados en secuencias diferenciadas, las mismas que pueden contener regiones que sean muy similares o que sean un motivo de secuencias similares dentro de un contexto mayor.

Este alineamiento permite encontrar similitudes que se pueden repetir, o puedan aparecer en distinto orden a lo largo de toda la secuencia.

En esencia los alineamientos locales buscan regiones entre las dos secuencias parecidas, aunque estas se encuentren rodeadas de zonas completamente diferentes.

[8] http://es.wikipedia.org/wiki/Algoritmo_Needleman-Wunsch

Una implementación de este alineamiento es el algoritmo de Smith-Waterman, el cual está basado en programación dinámica.

Cuando dos secuencias son suficientemente similares, no existe diferencia entre alineamientos globales y locales[3].

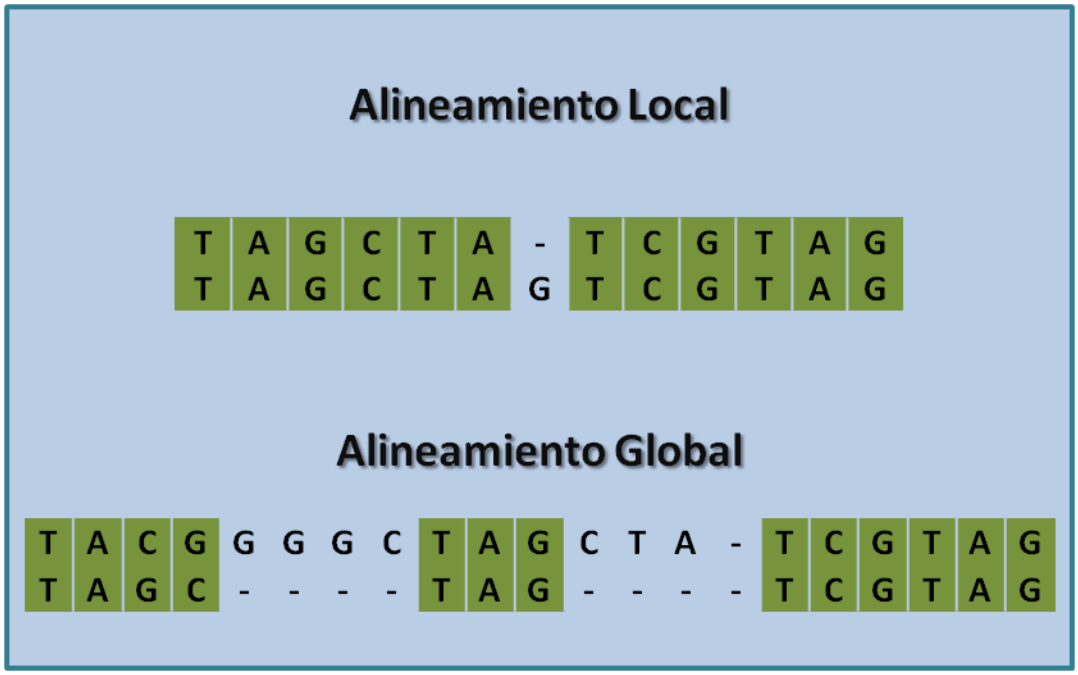


Figura 9 Comparación entre un alineamiento local y global

2.2. Algoritmo Smith-Waterman

El algoritmo de Smith-Waterman fue propuesto por Temple Smith y Michael Waterman en 1981. Es un método de comparación entre dos secuencias con el objetivo de identificar la mayor cantidad de segmentos similares. Está basado

en el uso de algoritmos de programación dinámica, lo cual garantiza que el alineamiento local encontrado es óptimo con respecto a un determinado sistema de puntuación utilizado.

El algoritmo usa una Matriz de puntuación en la cual se define a cada elemento de la matriz S_{ij} como la máxima puntuación obtenida del alineamiento entre el caracter correspondiente a la posición ***i*** de la secuencia ***X*** y el de la posición ***j*** correspondiente a la secuencia ***Y***. La relación de recurrencia estándar para S_{ij} donde h = hit, d = indel y r = replace es la siguiente:

$$S_{i,j} = \begin{cases} S_{i-1,j-1} + h & ; X(i) = Y(j) \\ \max(0,\ S_{i-1,j} - d,\ S_{i,j-1} - d,\ S_{i-1,j-1} - r) & ; \text{para cualquier otro caso} \end{cases}$$

Figura 10 Cálculo de los elementos de la Matriz S

Teniendo como condiciones iniciales:

$$S_{i,0} = S_{0,j} = 0 \qquad ; \text{para todo } i, j$$

El siguiente ejemplo toma como sistema de puntuación la propuesta del PhD. Robert Irving para detectar plagio hit = indel = replacement = 1 debido a que el algoritmo como tal castiga a los indels y replaces y premia a los hits.

Las secuencias a alinear son las siguientes:

X = "abcdefghij"
Y = "abcxdefghiymzj"

Aplicando la relación de recurrencia a cada elemento de ***S*** tenemos como resultado la siguiente matriz.

	a	b	c	x	d	e	f	g	h	i	y	m	z	j
a	**1**													
b		**2**	1											
c		1	**3**	**2**	1									
d			2	2	**3**	2	1							
e			1	1	2	**4**	3	2	1					
f					1	3	**5**	4	3	2	1			
g						2	4	**6**	5	4	3	2	1	
h						1	3	5	**7**	6	5	4	3	2
i							2	4	6	**8**	**7**	**6**	**5**	4
j							1	3	5	7	7	6	5	**6**

Figura 11 Vista de la matriz generada

El siguiente paso del algoritmo consiste en determinar la cantidad y lugar donde se van a insertar los espacios. Para ello se realiza un proceso de recorrido hacia

atrás, el cual comienza desde el último elemento de la matriz que se encuentra en la esquina inferior derecha y finaliza en el primer elemento de la matriz que se encuentra en la esquina superior izquierda. Para lograr este recorrido se determina el antecesor del elemento y así consecutivamente con el siguiente padre en cuestión de acorde al siguiente criterio:

Si	$S_{i,j} = 0$	Entonces	(i, j) no tiene antecesores
Si	$X(i) = Y(j)$	Entonces	(i, j) tiene su antecesor en $(i-1, j-1)$
De otro modo			
	(i, j) tiene su antecesor en $(p, q) \in \{(i-1, j), (i, j-1)\} / S_{i,j} = S_{p,q} - d$		
	$^{y}/_{o}\ (i, j)$ tiene su antecesor en $(i-1, j-1) / S_{i,j} = S_{i-1,j-1} - r$		

Figura 12 Cálculo del alineamiento entre dos secuencias

Hay que tener en cuenta que una posición no necesariamente tiene un antecesor único lo cual puede originar varias rutas. Si el recorrido toma un comportamiento horizontal esto indica que en la cadena se debe realizar una inserción de un espacio por cada antecesor que se encuentre en esta posición, mientras que si el antecesor se encuentra de manera diagonal esto puede representar un *"hit"* o un *"replace"* dependiendo de la coincidencia entre ***X(i)*** y ***Y(j)***. En la Figura 11 el recorrido hacia atrás está denotado en negrillas. El resultado del recorrido nos da el siguiente alineamiento:

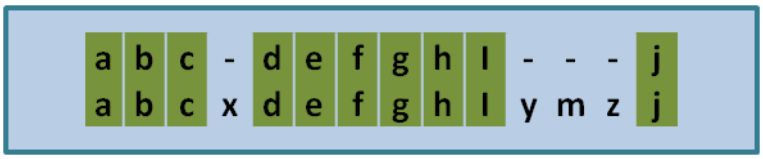


Figura 13 Resultado del alineamiento tomado de la Matriz

Teniendo como resultado final 10 hits, 4 indels y 0 replaces, Para determinar el puntaje del alineamiento aplicado se penaliza los indels y los replaces, obteniendo como resultado lo siguiente de acorde al sistema de puntuación elegido *(hit = replace= indel =1)* **10*1 – 4*1 – 0*1 = 6**, lo cual se lo puede comprobar con el último valor (esquina inferior derecha) de la matriz de puntuación.

2.3. Propuesta de Robert Irving para detección de plagios y colusiones

El algoritmo anteriormente citado muestra una puntuación determinada pero esta puntuación puede mejorar si se realizan cortes en la revisión de la cadena, de tal manera que la puntuación total mejore compuesta por cada uno de estos subconjuntos.

A continuación mostramos un ejemplo donde podemos notar que al realizar cortes a lo largo de la cadena podemos obtener una mejor puntuación.

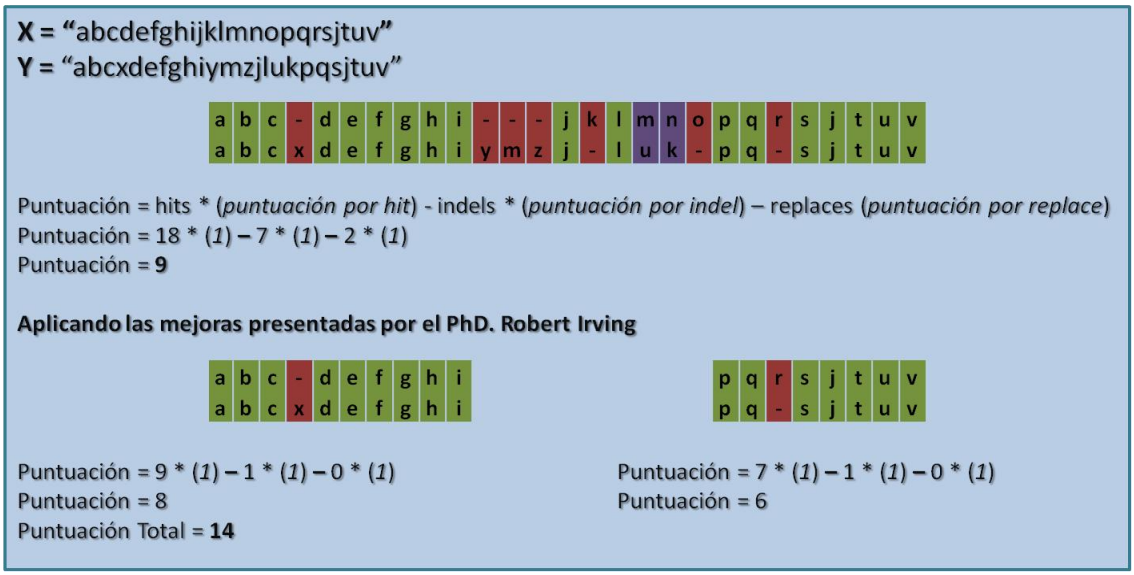


Figura 14 Aplicación de la revisión del algoritmo de Smith-Waterman

Para aplicar estos cortes a lo largo de la cadena se debe calcular una nueva matriz denominada M, la cual se la calcula de la siguiente manera.

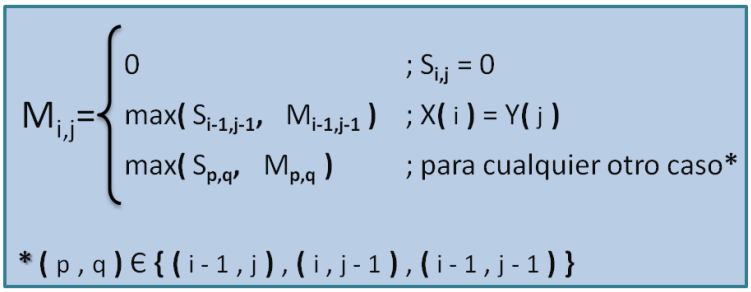


Figura 15 Cálculo de la Matriz M

Luego de haber calculado esta matriz se deben tomar las coincidencias más significativas, esto se logra definiendo un valor de umbral o "*threshold"* el cual sirve como base para determinar cuáles serán tomadas como coincidencias significativas, siendo 1 un valor estricto, esto implicaría que una coincidencia es considerada significativa.

CAPÍTULO 3

3. Tecnologías Aplicadas

3.1. Amazon EC2

Amazon Elastic Compute Cloud (*EC2 por sus siglas en inglés*) es un servicio tolerable a fallos y de pago por demanda, que provee capacidad de cómputo variable en la nube del Internet, está diseñado para hacer del procesamiento escalable de datos sobre la web, algo fácil de entender y adoptar para los desarrolladores.

Es una interfaz sencilla de servicio web fácil de configurar y adaptar a las necesidades del desarrollador. Proporciona un control completo de los recursos de procesamiento y permite ejecutar aplicaciones de proceso paralelo sobre la infraestructura física de la empresa.

Amazon EC2 reduce el tiempo necesario para obtener y reiniciar nuevas instancias de servidores virtuales (*AMIs por sus siglas en inglés*) a minutos,

permitiendo rápidamente escalar la capacidad de procesamiento en cuanto los requerimientos de la aplicación a desarrollarse así lo necesiten.

3.2. Amazon S3

Amazon Simple Storage Service (*Amazon S3 por sus siglas en inglés*) es un servicio de almacenamiento para Internet que provee mecanismos de autenticación, diseñado justamente por Amazon en paralelo con su servicio de procesamiento escalable (*EC2*), para permitir un fácil manejo de datos de alto tamaño (*desde 1 byte hasta 5 gigabytes por dato*). La localización física de estos datos puede configurarse manualmente por región para optimizar tiempos y costos de transferencia.

Este servicio provee una interfaz web simple que puede ser usada para almacenar u obtener una cantidad ilimitada de datos en cualquier momento desde cualquier lugar con acceso a Internet.

3.3. Hadoop

Hadoop, proyecto de la Apache Software Foundation, es una plataforma desarrollada en java, que implementa un paradigma computacional llamado

Map/Reduce (*inspirado en trabajos previos de Google*) en el que el proceso principal es replicado en diferentes máquinas (*llamadas nodos y cuya cantidad es configurable*) para procesar en paralelo pequeñas porciones parciales de un conjunto total de datos.

Además, su propia implementación de un Sistema de Archivos Distribuidos (***HDFS*** *por sus siglas en inglés*) permite un manejo distribuido de una cantidad, virtualmente ilimitada, de archivos de gran tamaño en diferentes máquinas.

El HDFS al igual que cualquier otro Sistema de Archivos posee el concepto general de administración de data sobre unidades de almacenamiento o bloques, con la diferencia principal del tamaño máximo de información que se puede leer o escribir sobre esta unidad.

Normalmente, en los Sistemas de Archivos comunes el tamaño de un bloque es de apenas unos cuantos kilobytes, mientras que en HDFS el tamaño por defecto es de 64 megabytes (tamaño que además puede ser configurable), característica por la cual es posible el manejo de archivos de gran tamaño.

Otra de las características que implementa Hadoop, y que será utilizado en la implementación del módulo propuesto, es su mecanismo de **Caché Distribuida**, que permite publicar un archivo a los nodos en los que se está ejecutando una aplicación de tal manera que la lectura de este archivo no se realizará desde el

HDFS por cada Map que se ejecute, sino desde memoria, optimizando así el ancho de banda en la red interna de la granja de clústeres de Amazon, y el tiempo de ejecución total [4].

3.4. Modelo de Programación Map/Reduce

Llamado así justamente por las dos fases principales en las que se divide, es un paradigma de programación que expresa un proceso grande como una secuencia de operaciones distribuidas sobre un conjunto de datos expresados como pares clave/valor. Este paradigma en Hadoop permite vincular virtualmente un grupo finito de máquinas o nodos, cuya cantidad puede ser definida por el usuario.

En la fase del "Map", el framework divide los datos de entrada en un gran número de fragmentos y asigna cada uno a una tarea "Map" distribuyendo éstas a su vez, a cada uno de los nodos en los que se procesará estos datos. En cada nodo, la tarea "Map" procesa pares clave/valor de su fragmento asignado y produce otro conjunto parcial clave/valor. Es decir, que para cada clave/valor de entrada (C, V), la tarea "Map" definida por el usuario invoca una función que transmuta este par en otro (C', V').

Después de esta fase, el framework ordena el conjunto de datos intermedios por clave, produciendo un conjunto de tuplas (C', V'*) en donde todos los valores asociados con una clave en particular, aparecen juntos. Además, éste divide el conjunto de tuplas en una serie de fragmentos de igual número que las tareas "Reduce".

En la fase "Reduce", se procesa el fragmento de tuplas (K', V'*), invocando una función definida por el usuario que transforma cada entrada en un conjunto de pares clave/valor (K, V). Una vez más, el framework se encarga de distribuir las tareas "Reduce" a los diferentes nodos y se ocupa de enviar los fragmentos intermedios a cada uno de ellos.

El paradigma de programación Map/Reduce y el Sistema de Archivos Distribuidos, en conjunto, fueron diseñados que las tareas en cada fase sean ejecutadas en un marco tolerante a fallos, es decir que, si un nodo falla en medio del proceso que esté realizando, la tarea asignada es redistribuida entre los nodos restantes si un nodo falla en medio del proceso que esté realizando, la tarea asignada es redistribuida entre los nodos restantes [5].

CAPÍTULO 4

4. Análisis e Implementación de la Solución

Considerando los fundamentos teóricos revisados en los capítulos anteriores procederemos a presentar los procesos que se llevan a cabo como solución del problema.

Como precedente tenemos un conjunto de directorios distribuidos jerárquicamente que a su vez contienen todas las tareas digitales enviadas por los estudiantes a un sistema de administración de cursos. Además se debe considerar que se está trabajando sobre un ambiente en el cual ya se encuentra instalado Hadoop con sus respectivas configuraciones en una distribución de Linux.

Considerando los precedentes antes mencionados hemos definidos los siguientes procesos como implementación de la solución:

1. Estandarizar el conjunto de tareas a un formato de texto plano.

2. Copiar este directorio generado con los archivos estandarizados al HDFS.
3. Aplicar el algoritmo de Smith-Watherman implementado en un ambiente distribuido.

Estos procesos los realizamos con el objetivo de encontrar los porcentajes de similitud entre un conjunto de tareas que representan las tareas de un paralelo en particular con respecto a las de un repositorio completo.

Figura 16 Procesos de la implementación de la solución

4.1. Proceso de transformación a texto plano

El proceso de transformación a texto plano consiste en tomar todos los archivos con las extensiones definidas en el alcance del proyecto (pptx, ppt, xls, xlsx, doc, docx, pdf, html, htm, txt, zip, rar) del conjunto de directorios y transformar el contenido del archivo original a texto plano evitando detalles de estilo, formato, gráficos, etc. Para este proceso se desarrolló una solución en Java que se va a ejecutar en primera instancia con una gran cantidad de datos, luego de esto ya tenemos un repositorio establecido de archivos en texto plano y solo se aplicaría

este proceso por demanda es decir a medida que se vaya incrementando el repositorio.

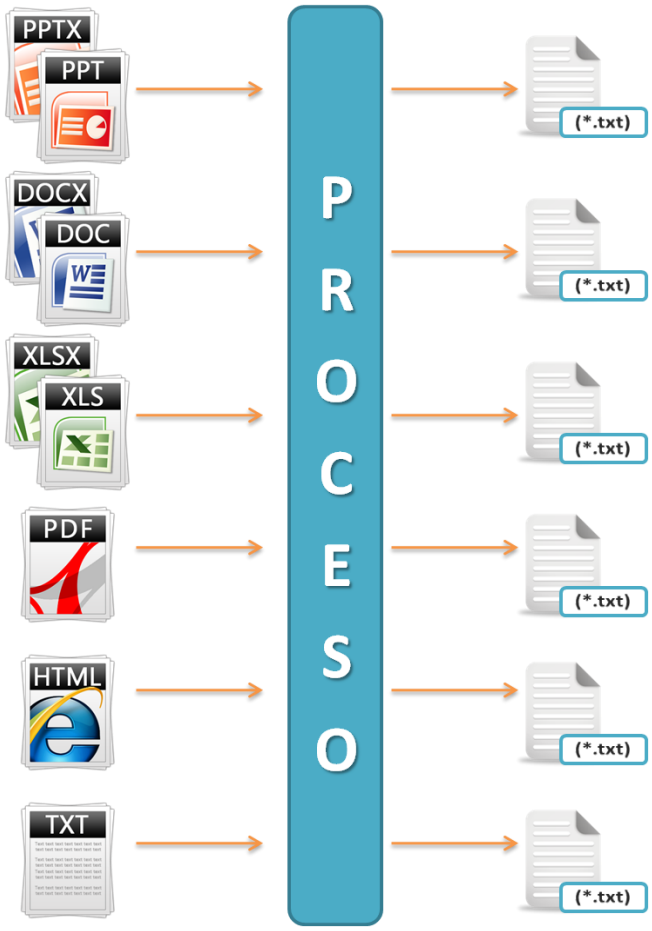


Figura 17 Proceso de Transformación a texto plano

Todos los archivos y directorios del repositorio de datos deben estar contenidos en una ruta origen la cual se la especifica en la variable **pathSource** de tipo String**.**

```
String pathSource = "/media/data/input/";
```

Código 1 Inclusión del directorio origen de datos

Utilizando la clase File de java procedemos a abrir el directorio origen de los datos y obtener todos los archivos y directorios que se encuentran en el primer nivel de la jerarquía. Este arreglo es uno de los parámetros que recibe el procedimiento **"convertListOfFiles"**, la misma que lleva a cabo el proceso de convertir los archivos a texto plano.

```
File folder = new File(pathSource);
File[] listOfFiles = folder.listFiles();
convertListOfFiles(listOfFiles,pathSource);
```

Código 2 Extracción de la lista de archivos

El procedimiento **"convertListOfFiles"** recorre cada uno de los archivos y directorios que se encuentran en el arreglo en busca de archivos que coincidan con el formato de (pptx, ppt, xls, xlsx, doc, docx, pdf, html, txt), este proceso lo realiza la función **"isSupportedFile"**, la cual recibe el nombre del archivo y retorna verdadero en caso de que sea un archivo soportado por el procedimiento, caso contrario retorna falso.

```
public static boolean isSupportedFile(String fileName);
```

Código 3 Función que determina si un archivo tiene un formato soportado

Luego de esto se procede a crear un archivo de texto plano el cual contiene como nombre un contador que se incrementa por cada archivo que ha sido transformado a texto plano. Simultáneamente en la variable “**buffer”**, la cual es una variable de tipo String y en esta se almacena la relación entre el número del archivo y la ruta completa de donde fue extraído ese archivo incluyendo el verdadero nombre del archivo.

```
for (int i = 0; i < listOfFiles.length; i++) {
  if (listOfFiles[i].isFile()) {
    try {
      fileName = listOfFiles[i].getName();
      if(isSupportedFile(fileName)){
        buffer += cont + "\t" + pathSource+fileName + "\n";
        File newFile = new File(pathResult + (cont++) + ".txt");
        FileWriter newFileWriter = new FileWriter(newFile);
        BufferedWriter out = new BufferedWriter(newFileWriter);
```

Código 4 Escritura del archivo base.txt

A continuación se muestra una imagen que ilustra el resultado de cada uno de los archivos procesados y una muestra del archivo “**base.txt”**.

Figura 18 Resultado de archivos procesados

```
134     /media/data/input/dlavayen/2006/i/poo/books/3e/cdinfo.html
135     /media/data/input/dlavayen/2006/i/poo/books/3e/index.html
136     /media/data/input/dlavayen/2006/i/poo/books/3e/bookTOC.pdf
137     /media/data/input/dlavayen/2006/i/poo/books/3e/errata.html
```

Figura 19 Muestra del archivo base.txt

Para transformar cada uno de los archivos a texto plano se usaron librerías que transforman los distintos formatos a texto plano. Para transformar los documentos de Microsoft usamos la librería **POI**[9], mientras que para los documentos pdf usamos la librería **pdfbox**[10].

```
1  if(fileName.endsWith(".doc") || fileName.endsWith(".docx"))
2    Text = convertDocToText(pathSource+fileName);
3  else if (fileName.endsWith(".xls") || fileName.endsWith(".xlsx"))
4    Text = convertExcelToText(pathSource+fileName);
5  else if (fileName.endsWith(".ppt") || fileName.endsWith(".pptx"))
6    Text = convertPowerPointToText(pathSource+fileName);
7  else if (fileName.endsWith(".pdf"))
8    Text = convertPDFToText(pathSource+fileName);
9  else if (fileName.endsWith(".txt"))
10   Text = convertTXTToText(pathSource+fileName);
11 else if (fileName.endsWith(".html") || fileName.endsWith(".htm"))
12   Text = convertTXTToText(pathSource+fileName);
```

Código 5 Llamadas a funciones que transforman a texto plano los diferentes formatos

En la variable **Text** se almacena el contenido del archivo procesado, para motivos de nuestro análisis las palabras tildadas y no tildadas al igual que las mayúsculas y minúsculas representan el mismo valor. Los stopwords, los signos de puntuación, los caracteres especiales, secuencias de escape y los errores en

[9] http://poi.apache.org/
[10] http://pdfbox.apache.org/

Excel (#REF!, #DIV/0!, etc.), son reemplazados por un espacio que luego serán omitidos por un solo espacio de tal manera que para el análisis en el texto final quedarán palabras separadas por un solo espacio. Para remplazar estos caracteres mencionados usamos una función denominada **replaceWords** que implementa lo antes mencionado.

```
public static String replaceWords(String Text){
  return Text.toLowerCase().replaceAll("á", "a").replaceAll("é", "e")
  .replaceAll("í", "i").replaceAll("ó", "o").replaceAll("ú", "u")
  .replaceAll("\\#....?.?", " ").replaceAll("[^a-zA-Z0-9]", " ")
  .replaceAll(" el "," ").replaceAll(" me "," ").replaceAll(" nos "," ")
  .replaceAll(" te "," ").replaceAll(" ante "," ").replaceAll(" en(tre)? "," ")
  .replaceAll(" con(tra)? "," ").replaceAll(" a(l)? "," ")
  .replaceAll(" l(o|a)s "," ").replaceAll(" de(l)? "," ")
  .replaceAll(" un((a|o)s?)? "," ").replaceAll(" mi((a|o)s?)? "," ")
  .replaceAll(" nuestr(o|a)s? "," ").replaceAll(" tuy(o|a)s? "," ")
  .replaceAll(" suy(o|a)s? "," ").replaceAll(" vuestr(o|a)s? "," ")
  .replaceAll(" bajo "," ").replaceAll(" cabe "," ").replaceAll(" desde "," ")
  .replaceAll(" durante "," ").replaceAll(" hacia "," ").replaceAll(" hasta "," ")
  .replaceAll(" mediante "," ").replaceAll(" para "," ").replaceAll(" por "," ")
  .replaceAll(" segun "," ").replaceAll(" sin "," ").replaceAll(" sobre "," ")
  .replaceAll(" tras "," ").replaceAll(" via "," ").replaceAll(" so "," ")
  .replaceAll(" os "," ").replaceAll(" +", " ").trim();
}
```

Código 6 Eliminación de caracteres innecesarios para el análisis

Luego de verificar si es un archivo soportado por el sistema se evalúa si es un paquete soportado, en este caso (rar o zip), para lo cual usamos los paquetes **zip** que se encuentran en **java.util**. Para el proceso de extraer los directorios y archivos de un .rar usamos la ejecución del comando **unrar**.

```
  if(fileName.endsWith(".zip"))
    unZipFile(fileName,pathSource);
  else if (fileName.endsWith(".rar"))
    unRarFile(fileName,pathSource);
```

Código 7 Archivos comprimidos soportados

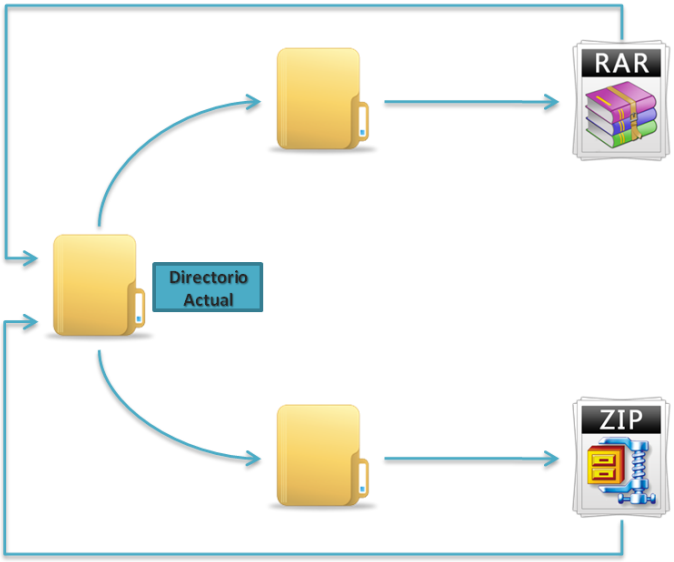


Figura 20 Ilustración de la extracción de archivos de directorios comprimidos

El proceso de transformación a texto plano se aplica recursivamente a los demás niveles jerárquicos de los directorios hasta cuando se hayan explorado todos los contenidos de los directorios.

Los archivos resultados del proceso de transformación se almacenan en un directorio que se lo define en el procedimiento **convertListOfFiles** específicamente en la variable **pathResult**, la cual es una variable de tipo String que almacena la ruta del directorio lo podemos observar en el Código 8 en la línea 19.

A continuación se presenta la implementación de la solución de transformación a texto plano.

```
public class Main {

  public static int cont = 0;
  public static String buffer = "";

  public static void main(String[] args) throws IOException {
    String pathSource = "/media/data/input/";
    File folder = new File(pathSource);
    File[] listOfFiles = folder.listFiles();
    convertListOfFiles(listOfFiles,pathSource);

    File base = new File("/base.txt");
    BufferedWriter out = new BufferedWriter(new FileWriter(base));
    out.write(buffer);
    out.close();
  }

  public static void convertListOfFiles(File[] listOfFiles, String pathSource){
    String pathResult = "/output/";
    String fileName;
    String Text = "";

    for (int i = 0; i < listOfFiles.length; i++) {
      if (listOfFiles[i].isFile()) {
        try {
          fileName = listOfFiles[i].getName();
          if(isSupportedFile(fileName)){
            buffer += cont + "\t" + pathSource+fileName + "\n";
            File newFile = new File(pathResult + (cont++) + ".txt");
            FileWriter newFileWriter = new FileWriter(newFile);
            BufferedWriter out = new BufferedWriter(newFileWriter);


            if (fileName.endsWith(".doc") || fileName.endsWith(".docx"))
              Text = convertDocToText(pathSource+fileName);
            else if (fileName.endsWith(".xls") || fileName.endsWith(".xlsx"))
              Text = convertExcelToText(pathSource+fileName);
            else if (fileName.endsWith(".ppt") || fileName.endsWith(".pptx"))
              Text = convertPowerPointToText(pathSource+fileName);
            else if (fileName.endsWith(".pdf"))
              Text = convertPDFToText(pathSource+fileName);
            else if (fileName.endsWith(".txt"))
              Text = convertTXTToText(pathSource+fileName);
            else if (fileName.endsWith(".html"))
              Text = convertTXTToHTML(pathSource+fileName);
            String temp = Text.toLowerCase().replaceAll("á", "a")
              .replaceAll("é", "e").replaceAll("í", "i")
              .replaceAll("ó", "o").replaceAll("ú", "u")
              .replaceAll("\\#....?.?"," ").replaceAll("[^a-zA-Z0-9]"," ")
              .replaceAll(" +"," ").trim();
            out.write(temp);
            out.close();
            newFileWriter.close();
            System.gc();
```

```
55            }
56            else if (fileName.endsWith(".zip"))
57              unZipFile(fileName,pathSource);
58            else if (fileName.endsWith(".rar"))
59              unRarFile(fileName,pathSource);
60          } catch (Exception ex) {
61            Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
62        }
63      }
64      else{
65        convertListOfFiles(
66        listOfFiles[i].listFiles() ,pathSource + listOfFiles[i].getName() + "/");
67      }
68    }
69  }
70
71
```

Código 8 Implementación del proceso de transformación a texto plano

4.2. Proceso de Copia de Archivos al HDFS

Luego de que el proceso de transformación a texto plano haya finalizado de estandarizar los archivos, los mismos se encuentran en un directorio del sistema de archivos del sistema operativo, para aplicar el modelo de programación Map Reduce es necesario que los archivos se encuentren en un sistema de archivos distribuidos como es el HDFS.

Para copiar el directorio del sistema de archivos local al sistema de archivos distribuidos se aplica el siguiente comando en la consola.

```
$ hadoop fs -put /output /user/lista
```

Figura 21 Comando para copier directories al HDFS

En este ejemplo mostrado en la Figura 21 se está copiando la carpeta **/output** del sistema de archivos local a la carpeta **/user/lista** que se encuentra en el sistema de archivos distribuido (HDFS).

4.3. Proceso de identificación de plagio

El proceso de identificación de plagio consiste en utilizar el procesamiento masivo de datos a través de Hadoop con la implementación del algoritmo de Smith-Waterman.

El algoritmo de Smith-Waterman esta implementado en la clase mapper, en donde el mapper va a recibir el archivo que este procesando Hadoop del repositorio alojado en el HDFS, luego extraerá de la cache del HDFS la tarea a analizar y aplicará el algoritmo generando como resultado el porcentaje de similitud entre esos archivos.

Visualmente podríamos apreciar el proceso Map/Reduce en la Figura 22, donde se observa cómo un solo archivo es tomado como base para las comparaciones sucesivas de la caché distribuida.

Dentro de la implementación del modelo Map Reduce hay que crear una clase **main**, en donde se configuran los parámetros necesarios para el funcionamiento del proceso distribuido, tales como la clase Map, Reduce, Main, los tipos de

parámetros que van a recibir tanto el Mapper como el Reducer; para nuestra implementación estos van a ser de tipo texto.

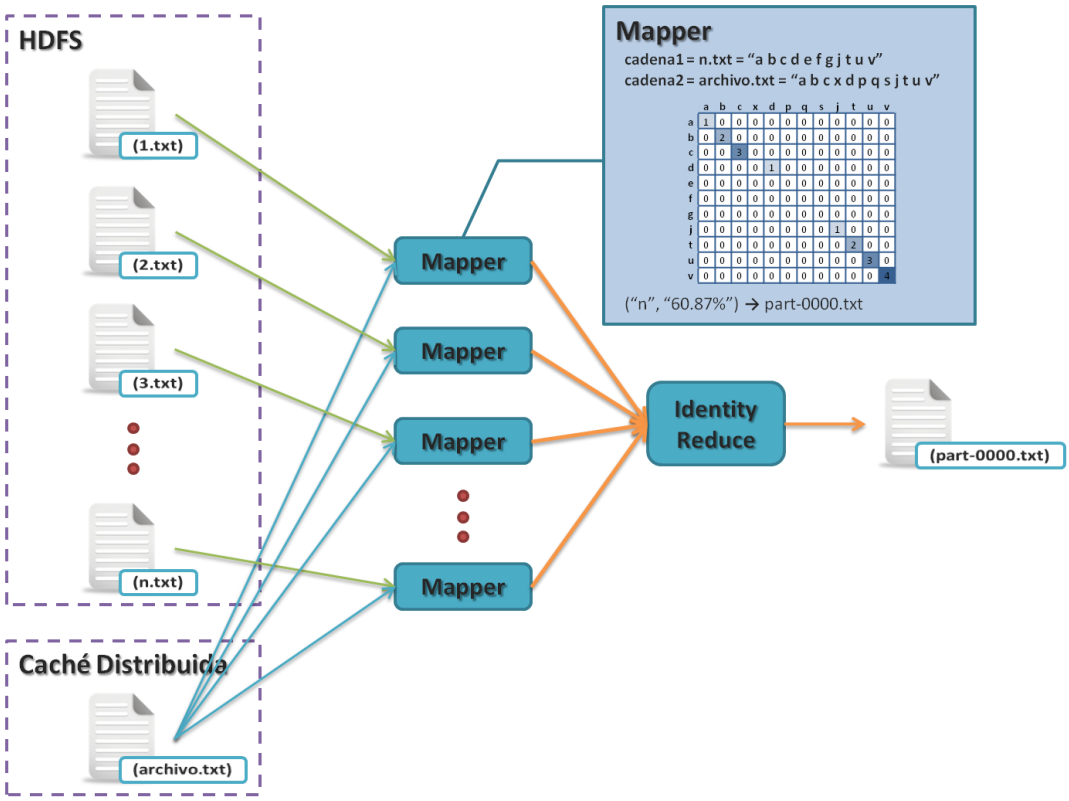


Figura 22 Modelo Map Reduce de la detección de plagio

En la clase **main** también se configura que archivo va a alojar nuestra cache distribuida, en nuestro caso va alojar el archivo que se está analizando contra los demás archivos del repositorio.

```
public class main {
    public static void main(String[] args) {
        try {
            JobClient client = new JobClient();
            JobConf conf = new JobConf(main.class);
            conf.setJobName("main");
            DistributedCache.releaseCache(URI.create("hdfs://ip-10-244-90-
            223.ec2.internal:9000/user/root/detectorArchivos/archivoNuevo"), conf);
            DistributedCache.purgeCache(conf);

            conf.setInputFormat(TextInputFormat.class);
            conf.setOutputFormat(TextOutputFormat.class);

            conf.setMapperClass(mapper.class);

            TextInputFormat.addInputPath(conf, new Path("detectorArchivos"));
            TextOutputFormat.setOutputPath(conf, new Path("detectorResultado"));

            conf.setNumReduceTasks(1);
            conf.setReducerClass(IdentityReducer.class);

            conf.setOutputKeyClass(Text.class);
            conf.setOutputValueClass(Text.class);

            client.setConf(conf);
            JobClient.runJob(conf);

            } catch (Exception e1) {
            e1.printStackTrace();
```

```
            }
30      }
31
```

Código 9 Implementación de la clase Main

El siguiente paso radica en la recuperación del archivo que se encuentra alojado en la cache distribuida. Este proceso se lo puede realizar en la clase Mapper la cual tiene un método denominado **configure** que permite recuperar el archivo de la cache distribuida y setear una variable denominada **cadena1** que es de tipo String, la cual va a tener como contenido el texto que se encuentre en el archivo recuperado de la cahce distribuida y esta variable va a estar disponible para todos los Mappers.

```
public void configure(JobConf conf){
  try{
    String cacheFile = "archivoNuevo";
    Path [] cacheFiles =
    DistributedCache.getLocalCacheFiles(conf);
    if(cacheFiles!= null && cacheFiles.length>0){
      for(Path cachePath : cacheFiles){
        if(cachePath.getName().equals(cacheFile)){
          FileReader fr = new FileReader(cachePath.toString())
          BufferedReader wordReader = new BufferedReader(fr);
          try{
            cadena1 = " " + wordReader.readLine().trim() + " ";
          }
          catch(Exception e){}
          finally{ wordReader.close();}
        }
      }
    }
```

Código 10 Extracción del archivo de la cache del HDFS

En la clase **mapper** definimos como variables nuestro sistema de puntuación, el cual se encuentra ponderado con un valor de 1 tanto para los **hits, indels, replaces** haciendo estricto el porcentaje de similitud entre los archivos comparados, además de este sistema definimos el **threshold** que nos denota las coincidencias significativas para lo cual hemos tomado el valor de 1, es decir el simple hecho de que coincida una palabra ya es una coincidencia significativa.

```
static int hit = 1, indel = 1, replacement = 1, threshold = 1;
```

Código 11 Definición de parámetros del algoritmo

La función map recibe como uno de sus argumentos una variable de tipo Reporter, la cual tiene como parte de su contenido un archivo extraído del HDFS

durante el procesamiento distribuido, de este archivo extraemos el contenido y lo almacenamos en la variable **“cadena2”**. El objetivo es crear dos arreglos de números a partir de **“cadena 1”** y **“cadena 2”** respectivamente en donde las palabras que se encuentren tanto en la primera cadena como en la segunda deberán tener el mismo valor numérico en ambos arreglos. Para implementar esta asociación se utiliza la ayuda de una estructura HashMap el cual almacena las palabras que hayan sido recorridas con un respectivo identificador numérico y en caso de que la palabra ya este en el HashMap se reemplaza por el valor correspondiente.

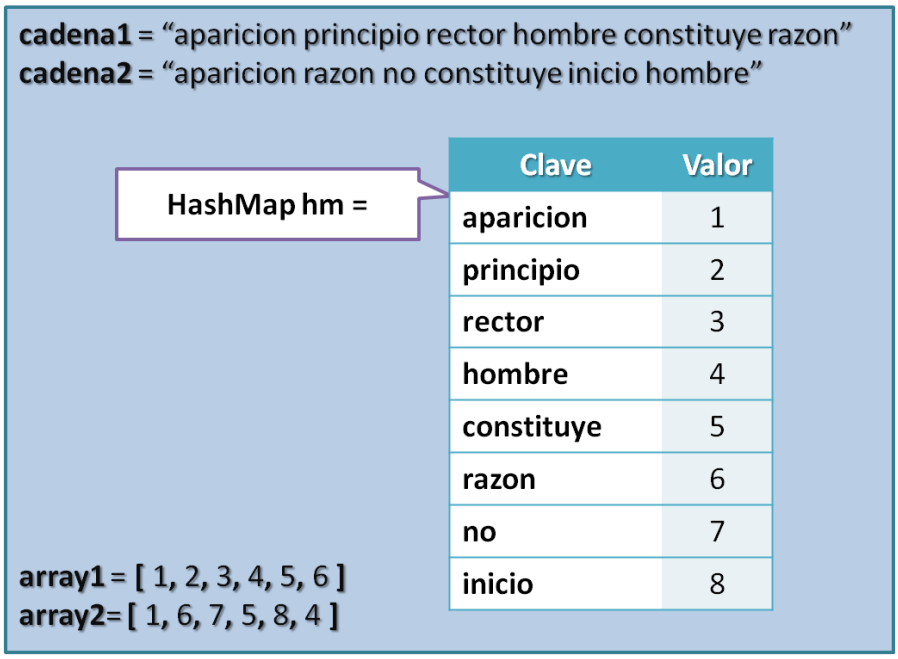


Figura 23 Ejemplo de conversión de números de cadena1 y cadena2

Para poder asociar la mayor cantidad de palabras con números tomamos en cuenta los números negativos y como inicio el mínimo numero negativo hasta el máximo entero positivo permitido.

```
public void map(WritableComparable offset, Writable valueFromOffset,
OutputCollector<Text,Text> responseToReducer, Reporter incommingFile)
throws IOException {
  FileSplit file = (FileSplit)incommingFile.getInputSplit();
  String fileName = file.getPath().getName();
  System.err.println(fileName);
  if(!fileName.equals("archivoNuevo") && valueFromOffset.toString().trim()
     .contains(" ")){
    String cadena1 =valueFromOffset.toString().trim();
    System.out.println(fileName);

    HashMap hm = new HashMap();

    StringTokenizer tokenizer1 = new StringTokenizer(cadena1);
    StringTokenizer tokenizer2 = new StringTokenizer(cadena2);
    int z = -2147483648;
    String token = "";

    while (tokenizer1.hasMoreTokens()) {
                    token = tokenizer1.nextToken();
                    if(!hm.containsKey(token)) hm.put(token, z++);
                    array1.add(hm.get(token));
                }
    while (tokenizer2.hasMoreTokens()) {
                    token = tokenizer2.nextToken();
                    if(!hm.containsKey(token)) hm.put(token, z++);
                    array2.add(hm.get(token));
                }
```

Código 12 Implementación de la conversión del texto a secuencias de números

Conociendo la cantidad de palabras en base a la dimensión de cada uno de los arreglos se crean la matriz S y la matriz M utilizando la librería **jama**[11] que permite un manejo optimo de matrices. Luego de este cálculo de ambas matrices usamos la forma revisada del algoritmo de Smith-Waterman la cual nos

[11] http://math.nist.gov/javanumerics/jama/

da como resultado las coincidencias significativas definidas por el *threshold* y almacenadas en la cadena C, de donde podemos obtener un resultado de cuantas coincidencias significativas existieron en esa comparación de un total de palabras y poder calcular en que porcentaje se parecen dos archivos.

```
int cadena1Length = array1.size();
int cadena2Length = array2.size();

Matrix S = inicializarMatriz(cadena1Length, cadena2Length);
Matrix M = inicializarMatriz(cadena1Length, cadena2Length);

String C = "";

for (int i = 0; i < cadena1Length; i++) {
  for (int j = 0; j < cadena2Length; j++) {
    //Calculando Sij
    if (array1.get(i).equals(array2.get(j))) {
      if (i == j && i == 0) {
        S.set(i,j,hit);
      } else {
        S.set(i,j, hit + valorAnterior(S, i, j, 1, 1));
      }
    } else {
      S.set(i,j, valorMaximo(
      valorAnterior(S, i, j, 1, 0) - indel,
      valorAnterior(S, i, j, 0, 1) - indel,
      valorAnterior(S, i, j, 1, 1) - replacement,
      0, 0, 0));
    }
    //Fin calculando Sij
    //Calculando Mij
    if (S.get(i,j) == 0) {
      M.get(i,j,0);
    }
    else {
      if (array1.get(i).equals(array2.get(j))) {
        M.set(i,j, valorMaximo(
        valorAnterior(S, i, j, 1, 1),
        valorAnterior(M, i, j, 1, 1),
        0, 0, 0, 0));
      } else {
        M.get(i,j, valorMaximo(
        valorAnterior(M, i, j, 1, 0),
        valorAnterior(M, i, j, 0, 1),
        valorAnterior(M, i, j, 1, 1),
        valorAnterior(S, i, j, 1, 0),
        valorAnterior(S, i, j, 0, 1),
        valorAnterior(S, i, j, 1, 1)));
      }
    }
    //Fin calculando Mij

```

```
    //Smith-Waterman revisado
    if (M.get(i,j) - S.get(i,j) >= threshold) {
      M.set(i,j,0);
      S.set(i,j,0);
    }
    if (S.get(i,j) >= threshold && S.get(i,j) > M.get(i,j]) {
      C += i + "-" + j + ";";
    }
    //Fin Smith-Waterman revisado
  }
}

String registro = "";
int count = 0;

for (int k = 0; k < C.split(";").length; k++) {
  try{
    String pair = C.split(";")[k];
    int i = Integer.parseInt(pair.split("-")[0]);
    int j = Integer.parseInt(pair.split("-")[1]);

    int aumentoI = 0;
    int aumentoJ = 0;

    if(i < array1.size()-1) aumentoI ++;
    if(j < array2.size()-1) aumentoJ ++;

    int iSiguiente = i + aumentoI;
    int jSiguiente = j + aumentoJ;

    if ((S.get(i,j)==threshold && S.get(iSiguiente,jSiguiente)>S.get(i,j) ||
    (S.get(i,j)>threshold) || (S.get(i,j)==threshold && i==0 && j==0) ||
    (S.get(i,j)==threshold && i==array1.size()-1 && j==array2.size()-1)) {
      if(!registro.contains(i + " ")){
        registro += i + " ";
        count++;
      }
    }
  }
  catch(Exception e){
    System.err.println(e.getMessage());
  }
}

System.out.println("Significants matches:" + count + " de:" + array1.size() + "
palabras. " + (float) count * 100 / array1.size() + "% de similitud");
```

Código 13 Implementación del algoritmo de Smith-Waterman

El *Reducer* se encarga de recolectar la información generada por los mappers y generar un archivo part con los resultados.

CAPÍTULO 5

5. Evaluación y Pruebas

En un ambiente de producción, el proceso de transformar los documentos a texto plano, explicado en el apartado 4.1, se realizaría de manera automática, inmediatamente después de que un usuario haya subido una tarea al Sistema de Administración de Cursos, por lo cual no va a ser objeto de prueba en este capítulo.

Es importante recalcar que el proceso de pruebas en esta sección nos permitió establecer y confirmar de manera empírica cómo los resultados de la detección de plagio con un gran volumen de datos se van comportando dependiendo del volumen de datos que se analiza y de la cantidad de nodos que se utiliza para el efecto, permitiéndonos sugerir una cantidad estimada de clústeres que serían necesarios para una cantidad determinada de archivos en un ambiente real de producción.

Para esta etapa se recreó un escenario con 86000 tareas que fueron transformados a archivos de texto plano cuyo peso total es de 1452 Mb ≈ 1.418 Gb simulando tareas enviadas en años, términos, cursos y/o paralelos anteriores; un conjunto de 35 de estas tareas representan los archivos enviados como respuesta a un deber de una clase, y un archivo sujeto al análisis que será denominado *archivoNuevo*.

Aproximadamente cada documento expresado en archivos de texto plano sin caracteres especiales y sin palabras vacías tienen en promedio 17.29 Kb y 1175 palabras. Los archivos que fueron utilizados son el resultado de la recopilación de tareas, apuntes y documentos reales de compañeros de clase de los últimos 4 años en la Universidad.

5.1. Preparando el escenario

Como se presentó en la Figura 19 del capítulo anterior, el archivo base.txt no es más que un conjunto de datos en formato clave-valor que representa la correspondencia de “código de archivo” “ubicación de archivo”. Para poder subir todos los archivos al S3 se procedió a generar un archivo en base a éste con un nuevo formato de la forma:

```
s3n://AWS_ACCESS_KEY_ID:AWS_SECRET_ACCESS_KEY@BUCKET/detectorArchivos/n.txt
```

Figura 24 Formato del archivo list

En donde ***n*** representa el código del archivo de cada línea.

Luego se realizaron cortes a este archivo de 86000 líneas en lotes de 500 para proceder a una copia distribuida de cada fragmento, ejecutando:

```
package main;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.DataInputStream;
import java.io.FileInputStream;
import java.io.FileWriter;
import java.io.InputStreamReader;
import java.util.Date;

public class main {
  public final static int TAMANIO_BUCKETS = 500;

  public static void main(String[] args) {
    try{
      int i = 0;
      int no_archivo = 0;
      String fileName = "";
      DataInputStream in = new DataInputStream(new FileInputStream("list"));
      BufferedWriter out = null;
      BufferedReader br = new BufferedReader(new InputStreamReader(in));
      String strLine;
      BufferedWriter out_time = new BufferedWriter(new FileWriter("time_copy"));

      Runtime.getRuntime().exec("mkdir detectorArchivos").waitFor();
      Runtime.getRuntime().exec("mkdir lista").waitFor();
      Runtime.getRuntime().exec("hadoop fs -mkdir /user/lista").waitFor();
      Runtime.getRuntime().exec("hadoop fs -mkdir
                        /user/root/detectorResultados").waitFor();
      Runtime.getRuntime().exec("hadoop fs -mkdir
                        /user/root/detectorArchivos").waitFor();

      while ((strLine = br.readLine()) != null)  {
        if(i%TAMANIO_BUCKETS==0){
          no_archivo = (i/TAMANIO_BUCKETS);
          fileName = "lista/list"+no_archivo+".ls";
          out_time.write("copia No. "+no_archivo+"\nInicio:"+
                         new Date().toString()+"\n");
          out = new BufferedWriter(new FileWriter(fileName));
        }

        out.write(strLine+"\n");

        if((++i)%TAMANIO_BUCKETS==0){
          out.close();
          Runtime.getRuntime().exec("/usr/bin/hadoop fs -put "+fileName+"
                            /user/lista/").waitFor();
          Runtime.getRuntime().exec("/usr/bin/hadoop distcp -f /user/"+fileName+"
                            /user/root/detectorArchivos").waitFor();
          Runtime.getRuntime().exec("/usr/bin/hadoop fs -rmr
```

```
                            /user/root/detectorArchivos/_*").waitFor();
          out_time.write("Fin:"+new Date().toString()+"\n");
        }
      }
      out.close();
      Runtime.getRuntime().exec("/usr/bin/hadoop fs -put "+fileName+"
                        /user/lista/").waitFor();
      Runtime.getRuntime().exec("/usr/bin/hadoop distcp -f /user/"+fileName+"
                        /user/root/detectorArchivos").waitFor();
      Runtime.getRuntime().exec("/usr/bin/hadoop fs -rmr
                        /user/root/detectorArchivos/_*").waitFor();
      out_time.write("Fin:"+new Date().toString()+"\n");
      out_time.close();
      in.close();

      Runtime.getRuntime().exec("/usr/bin/hadoop fs -get
                       /user/root/detectorArchivos/*"+"detectorArchivos").waitFor();
    }catch (Exception e){
      System.err.println("Error: " + e.getMessage());
    }
  }
}
```

Código 14 Implementación utilizada para realizar la copia distribuida por lotes

Esta copia distribuida en el S3 se la realizó con el fin de poder tener todo nuestro volumen de datos en los equipos físicos de Amazon para luego poderlos tomar en cualquier momento para las siguientes pruebas. Como se explicaba en líneas anteriores, en un ambiente de producción esta copia distribuida por lotes no sería necesaria, ya que después de convertir un documento en archivo plano, éste pasaría automáticamente al bucket determinado en S3.

Una vez levantado el nodo principal en donde correría nuestra aplicación, se revisa cual es el DNS configurado que le fue asignado a la máquina virtual, en el archivo hadoop-site.xml ubicado en /etc/hadoop/conf; para poder copiarlo y editar la ruta de la cual se copiaría el *archivoNuevo* a la caché distribuida.

En la clase principal y custodia del procesamiento en general de nuestra aplicación (*la clase main*), se realizaron los cambios pertinentes para simular el escenario planteado, tomando en cuenta que se realizarían por escenario 3 pruebas, para así tomar un valor promedio entre los resultados.

```
1  package proyecto;
2
3  import java.io.BufferedWriter;
4  import java.io.FileWriter;
5  import java.lang.reflect.Array;
6  import java.net.URI;
7  import java.util.ArrayList;
8  import java.util.Date;
9
10 import org.apache.hadoop.filecache.DistributedCache;
11 import org.apache.hadoop.fs.Path;
12 import org.apache.hadoop.io.Text;
13 import org.apache.hadoop.mapred.JobClient;
14 import org.apache.hadoop.mapred.JobConf;
15 import org.apache.hadoop.mapred.TextInputFormat;
16 import org.apache.hadoop.mapred.TextOutputFormat;
17 import org.apache.hadoop.mapred.lib.IdentityReducer;
18
19 public class main {
20   public final static int TOTAL_ARCHIVOS = 35;
21   public final static int TOTAL_PRUEBAS = 3;
22   public static void main(String[] args) {
23     try{
24       int pruebaN = 0;
25       BufferedWriter out = new BufferedWriter(new FileWriter("time_prueba_1"));
26
27       while(pruebaN < TOTAL_PRUEBAS){
28         int nArchivo = 0;
29         out.write("Prueba "+pruebaN+++"\n"+new Date().toString()+"\n");
30         while(nArchivo<TOTAL_ARCHIVOS){
31           try {
32             JobClient client = new JobClient();
33             JobConf conf = new JobConf(main.class);
34
35             conf.setJobName("main");
36
37             Runtime.getRuntime().exec("/usr/bin/hadoop fs -rmr
38                               /user/root/detectorResultado").waitFor();
39             Runtime.getRuntime().exec("rm /root/detectorResultado"+ nArchivo
40                               + ".txt").waitFor();
41             Runtime.getRuntime().exec("/usr/bin/hadoop fs -rm
42                               /user/root/detectorArchivos/archivoNuevo")
43                               .waitFor();
44             Runtime.getRuntime().exec("/usr/bin/hadoop fs -cp
45                               /user/root/detectorArchivos/archivoNuevo"+
46                               nArchivo + "/user/root/detectorArchivos/archivoNuevo")
47                               .waitFor();
48
```

```
49              DistributedCache.releaseCache(URI.create("hdfs://ip-10-244-90-
50              223.ec2.internal:9000/user/root/detectorArchivos/archivoNuevo"), conf);
51              DistributedCache.purgeCache(conf);
52              DistributedCache.addCacheFile(URI.create("hdfs://ip-10-244-90-
53              223.ec2.internal:9000/user/root/detectorArchivos/archivoNuevo"), conf);
54
55              conf.setInputFormat(TextInputFormat.class);
56              conf.setOutputFormat(TextOutputFormat.class);
57
58              conf.setNumMapTasks(2);
59              conf.setMapperClass(mapper.class);
60
61              TextInputFormat.addInputPath(conf, new Path("detectorArchivos"));
62              TextOutputFormat.setOutputPath(conf, new Path("detectorResultado"));
63
64              conf.setNumReduceTasks(1);
65              conf.setReducerClass(IdentityReducer.class);
66
67              conf.setOutputKeyClass(Text.class);
68              conf.setOutputValueClass(Text.class);
69
70              client.setConf(conf);
71              try {
72                JobClient.runJob(conf);
73                Runtime.getRuntime().exec("/usr/bin/hadoop fs -get
74                                          /user/root/detectorResultado/part-00000
75                                          /root/detectorResultado" + nArchivo+++
76 ".txt")
77                                          .waitFor();
78              } catch (Exception e) {
79                e.printStackTrace();
80              }
81            } catch (Exception e1) {
82              // TODO Auto-generated catch block
83              e1.printStackTrace();
84            }
85          }
86          out.write(new Date().toString()+"\n");
87        }
88        out.close();
89      }
90      catch (Exception e1) {
91        // TODO Auto-generated catch block
92        e1.printStackTrace();
93      }
94    }
  }
```

Código 15 Implementación utilizada para simular un entorno de producción

5.2. Pruebas de eficacia

Para ilustrar paso a paso el procedimiento realizado, tomaremos un par de archivos de nuestro conjunto de datos, presentados al final del documento como Anexo 1 y

Anexo 2; aunque sus nombres de archivos son originalmente “Trabajo de Ecología.doc” y “Deber de Ecología y educación Ambiental.pdf” los llamaremos por simplicidad: original y copia (en ese orden respectivamente).

Como se puede ver en la copia, los cambios obedecen a modificaciones en formato, tipo de texto, inserción de imágenes, cabeceras, pies de página y cambio en el formato de presentación (*pdf*). Además los párrafos fueron dispuestos en un orden diferente con la mínima modificación en su contenido.

Después del proceso de transformar los documentos a texto plano, parte de los resultados son los siguientes:

```
trabajo ecologia y educacion ambiental
diego lavayen alarcon paralelo 37
imagenes presentadas pagina hyperlink
http www equilibrioazul org www
equilibrioazul org sin duda alguna
pueden motivarme comentar sobre vida que
poco poco se esta perdiendo gracias
imprudencia e ignorancia hombre tantas
morenas y peces muertos aletas tortugas
regadas suelo marino tantos tiburones
blancos incapaces atacar a un humano
muertos solo capricho humano y solo esto
representar brutal matanza...
```

```
ecologia y educacion ambiental trabajo
perteneciente eduardo cruz ramirez
imagenes presentadas en hyperlink http
www equilibrioazul org www
equilibrioazul org pueden motivarme
comentar sobre vida que poco poco se
esta perdiendo gracias imprudencia e
ignorancia hombre tantos peces muertos
aletas tortugas regadas suelo marino
tantos tiburones blancos muertos solo
capricho humano y solo esto representar
la brutal matanza que dia dia se comete
no solo...
```

Figura 26 Parte inicial del resultado del archivo original en texto plano

Figura 25 Parte inicial del resultado del archivo copia en texto plano

Se puede observar como en los archivos planos se discrimina por completo cambios puntuales en el contenido, como por ejemplo: poner en mayúscula cierto texto, capitalizar y/o tildar palabras, signos de puntuación, etc.

Como se explicó anteriormente, en la fase Map los contenidos de los archivos a analizarse en cada instancia son convertidos automáticamente a un vector de números que representan de manera única a cada palabra.

Después de haber preparado este par de vectores, se realizar el algoritmo mejorado de Smith-Waterman (explicado ampliamente en los apartados 2.2 y 2.3) y como resultado obtenemos la matriz de puntos *S* modificada que, por cuyo tamaño, lo hemos simplificado a una representación gráfica, en la que cada palabra similar entre el documento original y la copia es representada por un punto negro.

Se puede apreciar en la Figura 27 cómo estos puntos forman diagonales segmentadas que representan coincidencias secuenciales de las palabras en ambos documentos.

La razón por la que dichas diagonales no son continuas y no se encuentran totalmente alineadas es debida a que las coincidencias existen sólo en ciertas secciones de ambos documentos, secciones que no necesariamente están presentes a la misma altura en los mismos (*debido al movimiento de los párrafos en la copia*).

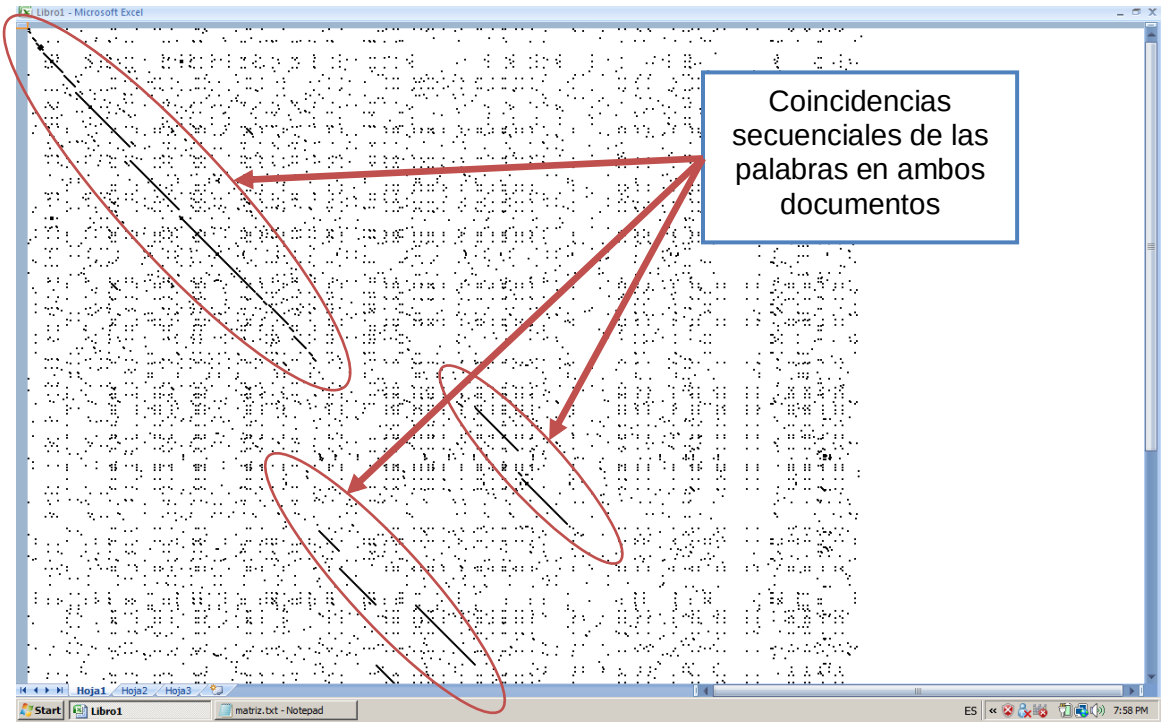


Figura 27 Representación gráfica de la matriz S en la comparación del Anexo 1 y 2

Al finalizar este ejemplo el sistema retornó un resultado de 87.86 %, indicando que de las 1120 palabras del documento original, 984 coinciden con el

documento copia no sólo en ocurrencia sino también en una secuencia determinada.

Partiendo de la idea principal que el algoritmo compara dos documentos cualesquiera, analizaremos dos archivos en texto plano en un escenario diferente, comparando las similitudes entre los textos de la Santa Biblia: Mateo 11_5 – 11_{11} y Lucas 7_{22} – 7_{28} propuestos en el presente como Anexo 3 y Anexo 4 como archivos html provistos por http://www.iglesia.net/biblia/.

```
biblia lucas haga clic capitulo que
desee ir 1 2 3 4 5 6 7 8 9 10 11 12 13
14 15 16 17 18 19 20 21 22 23 24
capitulo 17 22 y respondiendo jesus dijo
id haced saber juan que habeis visto y
oido ciegosven cojos andan leprosos son
limpiados ...
```

Figura 29 Parte inicial del archivo lucas7.22-7.28.html

```
biblia san mateo haga clic capitulo que
desee ir 1 2 3 4 5 6 7 8 9 10 11 12 13
14 15 16 17 18 19 20 21 22 23 24
capitulo 111 5 ciegos ven cojos andan
leprosos son limpiados sordos oyen
muertos son resucitados y a pobres es
anunciado evangelio 11 6 y ...
```

Figura 28 Parte inicial del archivo mateo11.5-11.11.html

La representación Gráfica de la Matriz S modificada es:

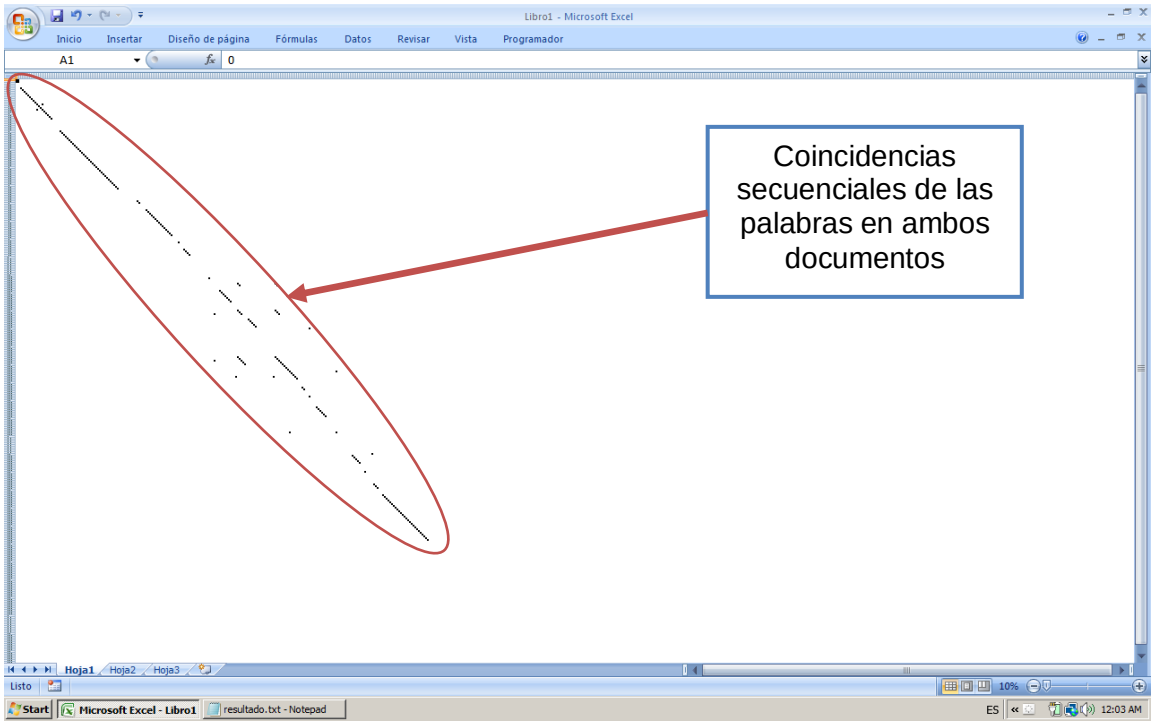


Figura 30 Representación gráfica de la matriz S en la comparación del Anexo 3 y 4

Así como en la Figura 30, podemos observa en esta, diagonales no continuas y levemente desalineadas, debido a los cortos segmentos en donde se encontró similitud entre ambos documentos.

Al finalizar este segundo ejemplo el sistema presenta una similitud del 66.66 %, indicando que de las 231 palabras del documento correspondiente al fragmento del libro de Lucas, 172 coinciden con el documento del fragmento de Mateo no sólo en ocurrencia sino también en una secuencia determinada.

En general, en un ambiente distribuido utilizando las técnicas de procesamiento masivo y escalable de datos sobre la plataforma Hadoop, el algoritmo implementado en esta solución presenta resultados favorables en sus pruebas, dando como resultado un archivo llamado part-00000 que muestra de manera secuencial, el código del archivo al cual el archivo "*archivoNuevo*" se parece, con un indicador (*porcentaje*) a su lado, que representa la probabilidad de certeza de plagio entre dichos archivos.

El archivo en mención presenta el siguiente formato:

```
Archivo part-0000

523    92.45
126    57.11
```

Figura 31 Archivo Resultado de la prueba

La muestra presentada en la Figura 31 indica que el *archivoNuevo* (*archivo que se está analizando en una iteración*) se parece al archivo cuyo código es 523 en un 92.45% según el algoritmo implementado, así mismo hace mención al archivo con código 126 con el que tiene una semejanza del 57.11 %.

Una implementación no sujeta a la presente, podría consumir este archivo para adjuntarse a una base de datos y presentar dicho resultado vía web, para que así los profesores tengan de manera eficiente y eficaz una percepción de las tareas que sus estudiantes han enviado.

5.3. Preparando el escenario de pruebas

Como se presentó en la Figura 19 del capítulo anterior, el archivo base.txt no es más que un conjunto de datos en formato clave-valor que representa la correspondencia de “código de archivo” “ubicación de archivo”. Para poder subir todos los archivos al S3 se procedió a generar un archivo en base a éste con un nuevo formato de la forma:

```
s3n://AWS_ACCESS_KEY_ID:AWS_SECRET_ACCESS_KEY@BUCKET/detectorArchivos/n.txt
```

Figura 32 Formato del archivo list

En donde ***n*** representa el código del archivo que en cada línea

Luego se realizaron cortes a este archivo de 86000 líneas en lotes de 500 para proceder a una copia distribuida de cada fragmento, ejecutando:

```
1  package main;
2
3  import java.io.BufferedReader;
4  import java.io.BufferedWriter;
5  import java.io.DataInputStream;
6  import java.io.FileInputStream;
7  import java.io.FileWriter;
8  import java.io.InputStreamReader;
9  import java.util.Date;
10
11 public class main {
12   public final static int TAMANIO_BUCKETS = 500;
13
14   public static void main(String[] args) {
15     try{
16       int i = 0;
17       int no_archivo = 0;
18       String fileName = "";
19       DataInputStream in = new DataInputStream(new FileInputStream("list"));
20       BufferedWriter out = null;
21       BufferedReader br = new BufferedReader(new InputStreamReader(in));
22       String strLine;
23       BufferedWriter out_time = new BufferedWriter(new FileWriter("time_copy"));
24
25       Runtime.getRuntime().exec("mkdir detectorArchivos").waitFor();
```

```
26        Runtime.getRuntime().exec("mkdir lista").waitFor();
27        Runtime.getRuntime().exec("hadoop fs -mkdir /user/lista").waitFor();
28        Runtime.getRuntime().exec("hadoop fs -mkdir
29                          /user/root/detectorResultados").waitFor();
30        Runtime.getRuntime().exec("hadoop fs -mkdir
31                          /user/root/detectorArchivos").waitFor();
32
33        while ((strLine = br.readLine()) != null)   {
34          if(i%TAMANIO_BUCKETS==0){
35            no_archivo = (i/TAMANIO_BUCKETS);
36            fileName = "lista/list"+no_archivo+".ls";
37            out_time.write("copia No. "+no_archivo+"\nInicio:"+
38                           new Date().toString()+"\n");
39            out = new BufferedWriter(new FileWriter(fileName));
40          }
41
42          out.write(strLine+"\n");
43
44          if((++i)%TAMANIO_BUCKETS==0){
45            out.close();
46            Runtime.getRuntime().exec("/usr/bin/hadoop fs -put "+fileName+"
47                              /user/lista/").waitFor();
48            Runtime.getRuntime().exec("/usr/bin/hadoop distcp -f /user/"+fileName+"
49                              /user/root/detectorArchivos").waitFor();
50            Runtime.getRuntime().exec("/usr/bin/hadoop fs -rmr
51                              /user/root/detectorArchivos/_*").waitFor();
52            out_time.write("Fin:"+new Date().toString()+"\n");
53          }
54        }
55        out.close();
56        Runtime.getRuntime().exec("/usr/bin/hadoop fs -put "+fileName+"
57                          /user/lista/").waitFor();
58        Runtime.getRuntime().exec("/usr/bin/hadoop distcp -f /user/"+fileName+"
59                          /user/root/detectorArchivos").waitFor();
60        Runtime.getRuntime().exec("/usr/bin/hadoop fs -rmr
61                          /user/root/detectorArchivos/_*").waitFor();
62        out_time.write("Fin:"+new Date().toString()+"\n");
63        out_time.close();
64        in.close();
65
66        Runtime.getRuntime().exec("/usr/bin/hadoop fs -get
67                         /user/root/detectorArchivos/*"+"detectorArchivos").waitFor();
68      }catch (Exception e){
69        System.err.println("Error: " + e.getMessage());
70      }
71    }
72  }
73
74
```

Código 16 Implementación utilizada para realizar la copia distribuida por lotes

Esta copia distribuida en el S3 se la realizó con el fin de poder tener todo nuestro volumen de datos en los equipos físicos de Amazon para luego poderlos tomar en cualquier momento para las pruebas siguientes.

Como se explicaba en líneas anteriores, en un ambiente de producción esta copia distribuida por lotes no sería necesaria, ya que después de convertir un documento en archivo de texto plano, éste pasaría automáticamente al bucket determinado en S3.

Una vez levantado el nodo principal en donde correría nuestra aplicación, se revisa cual es el DNS configurado que le fue asignado a la máquina virtual, en el archivo hadoop-site.xml ubicado en `/etc/hadoop/conf`; para poder copiarlo y editar la ruta de la cual se copiaría el *archivoNuevo* a la caché distribuida.

En la clase principal y custodia del procesamiento en general de nuestra aplicación (*la clase main*), se realizaron los cambios pertinentes para simular el escenario planteado, tomando en cuenta que se realizarían por escenario 3 pruebas, para así tomar un valor promedio entre los resultados.

```
1  package proyecto;
2
3  import java.io.BufferedWriter;
4  import java.io.FileWriter;
5  import java.lang.reflect.Array;
6  import java.net.URI;
7  import java.util.ArrayList;
8  import java.util.Date;
9
10 import org.apache.hadoop.filecache.DistributedCache;
11 import org.apache.hadoop.fs.Path;
```

```
12 import org.apache.hadoop.io.Text;
13 import org.apache.hadoop.mapred.JobClient;
14 import org.apache.hadoop.mapred.JobConf;
15 import org.apache.hadoop.mapred.TextInputFormat;
16 import org.apache.hadoop.mapred.TextOutputFormat;
17 import org.apache.hadoop.mapred.lib.IdentityReducer;
18
19 public class main {
20   public final static int TOTAL_ARCHIVOS = 35;
21   public final static int TOTAL_PRUEBAS = 3;
22   public static void main(String[] args) {
23     try{
24       int pruebaN = 0;
25       BufferedWriter out = new BufferedWriter(new FileWriter("time_prueba_1"));
26
27       while(pruebaN < TOTAL_PRUEBAS){
28         int nArchivo = 0;
29         out.write("Prueba "+pruebaN+++"\n"+new Date().toString()+"\n");
30         while(nArchivo<TOTAL_ARCHIVOS){
31           try {
32             JobClient client = new JobClient();
33             JobConf conf = new JobConf(main.class);
34
35             conf.setJobName("main");
36
37             Runtime.getRuntime().exec("/usr/bin/hadoop fs -rmr
38                             /user/root/detectorResultado").waitFor();
39             Runtime.getRuntime().exec("rm /root/detectorResultado"+ nArchivo
40                             + ".txt").waitFor();
41             Runtime.getRuntime().exec("/usr/bin/hadoop fs -rm
42                             /user/root/detectorArchivos/archivoNuevo")
43                             .waitFor();
44             Runtime.getRuntime().exec("/usr/bin/hadoop fs -cp
45                             /user/root/detectorArchivos/archivoNuevo"+
46                             nArchivo + "/user/root/detectorArchivos/archivoNuevo")
47                             .waitFor();
48
49             DistributedCache.releaseCache(URI.create("hdfs://ip-10-244-90-
50             223.ec2.internal:9000/user/root/detectorArchivos/archivoNuevo"), conf);
51             DistributedCache.purgeCache(conf);
52             DistributedCache.addCacheFile(URI.create("hdfs://ip-10-244-90-
53             223.ec2.internal:9000/user/root/detectorArchivos/archivoNuevo"), conf);
54
55             conf.setInputFormat(TextInputFormat.class);
56             conf.setOutputFormat(TextOutputFormat.class);
57
58             conf.setNumMapTasks(2);
59             conf.setMapperClass(mapper.class);
60
61             TextInputFormat.addInputPath(conf, new Path("detectorArchivos"));
62             TextOutputFormat.setOutputPath(conf, new Path("detectorResultado"));
63
64             conf.setNumReduceTasks(1);
65             conf.setReducerClass(IdentityReducer.class);
66
67             conf.setOutputKeyClass(Text.class);
68             conf.setOutputValueClass(Text.class);
69
70             client.setConf(conf);
71             try {
72               JobClient.runJob(conf);
```

```
73                Runtime.getRuntime().exec("/usr/bin/hadoop fs -get
74                                          /user/root/detectorResultado/part-00000
75                                          /root/detectorResultado" + nArchivo+++
76 ".txt")
77                                          .waitFor();
78              } catch (Exception e) {
79                e.printStackTrace();
80              }
81            } catch (Exception e1) {
82              // TODO Auto-generated catch block
83              e1.printStackTrace();
84            }
85          }
86          out.write(new Date().toString()+"\n");
87        }
88        out.close();
89      }
90      catch (Exception e1) {
91        // TODO Auto-generated catch block
92        e1.printStackTrace();
93      }
94    }
  }
```

Código 17 Implementación utilizada para simular un entorno de producción

5.4. Evaluación de rendimiento

Después de las pruebas y resultados obtenidos y gracias a los datos de tiempo que se tomaron antes y después de cada procesamiento en clase principal o main que gobierna las fases Map/Reduce de todo el proceso Hadoop (Código 17), tabulamos los tiempos resultantes en cada iteración, resultando las siguientes tablas y gráficas comparativas:

5.4.1. Tiempo vs. Nodos

Para el primer grupo de pruebas, se estableció constante la cantidad de archivos (*volumen de datos*), variando la cantidad de nodos utilizados por prueba.

	2 Nodos	6 Nodos	10 Nodos	20 Nodos
1era. Prueba	1:53:49	1:03:34	0:47:59	0:48:53
2da. Prueba	1:46:26	1:08:31	0:52:51	0:34:07
3era. Prueba	2:10:11	1:12:49	0:45:43	0:38:31
Media	1:56:49	1:08:18	0:48:51	0:40:30

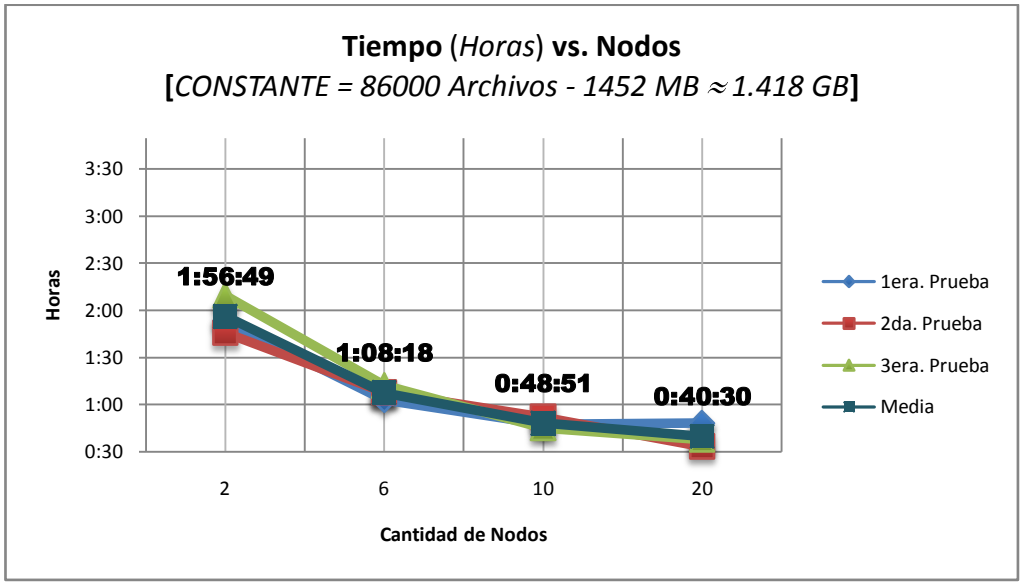


Figura 33 Resultados del primer grupo de pruebas (Tiempo vs. Nodos)

Observamos en la misma como el tiempo requerido para el procesamiento de una cantidad constante de datos (según las pruebas realizadas) se ve reducido con el aumento de nodos asociados a cada procesamiento.

Podemos ver cómo la media se va comportando de manera poco variable a partir del uso de aproximadamente 8 clústeres.

Al finalizar la prueba 1 observamos cómo el tiempo tomado aumenta en referencia al resto de pruebas con 20 Nodos, esta anomalía se presenta debido

a diversos aspectos como: el tráfico en la red interna en las granjas de clusterés de Amazon, la tolerancia a fallos (*caída inadvertida de un nodo y su inmediata reposición*), la demanda de los servicios de Amazon, entre otros…

Inicialmente es evidente cómo dos clústeres no son suficientes para la cantidad de datos que se están analizando, ya que aproximadamente 2 horas de procesamiento para 1.418Gb de datos es considerablemente mucho tiempo.
Ya cercanos a los 8 clústeres levantados se puede apreciar como la pendiente de la curva se va reduciendo con miras a permanecer poco variable en las siguientes pruebas.

5.4.2. Tiempo vs. Volumen de datos

Para este grupo de pruebas se estableció constante la cantidad de nodos utilizados para el procesamiento, manteniendo variable el volumen de datos para cada prueba.

	221MB 25000 Archivos	**634MB** 45000 Archivos	**1126MB** 65000 Archivos	**1452MB** 86000 Archivos
1era. Prueba	0:08:16	0:13:06	0:26:37	0:49:47
2da. Prueba	0:11:15	0:16:36	0:33:35	0:46:16
3era. Prueba	0:09:09	0:14:51	0:29:11	0:45:17
Media	0:09:33	0:14:51	0:29:48	0:47:07

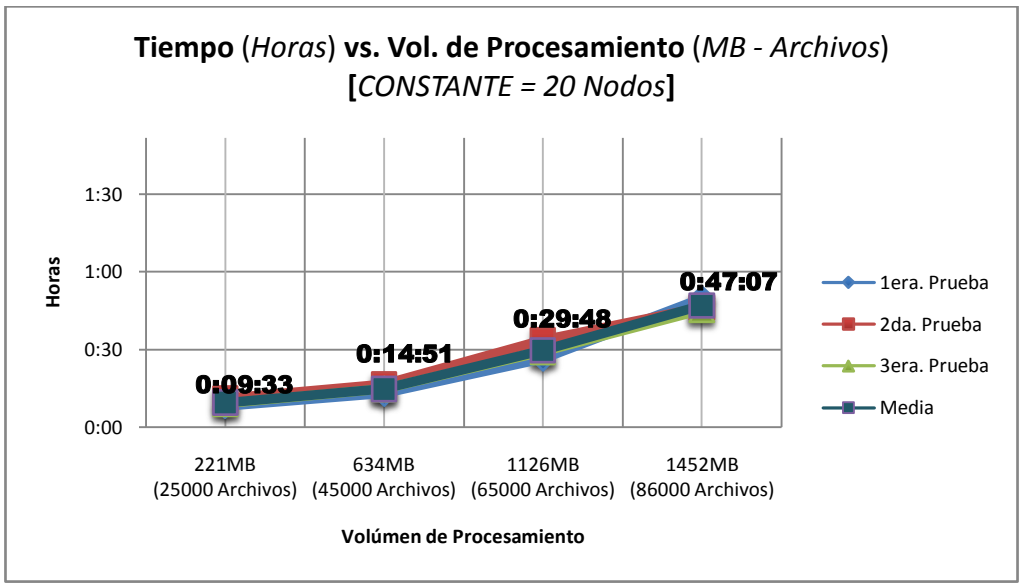


Figura 34 Resultado del segundo grupo de pruebas (Tiempo vs. Volumen de Datos)

Es evidente y comprobado con esta gráfica, que, a medida que aumenta el número de archivos analizados, considerando una cantidad constante de nodos levantados (para este caso 20 clústeres), el tiempo requerido se acrecienta proporcionalmente; llegando a un máximo local, para estas pruebas, de 47 minutos y 7 segundos. Es decir que, con un conjunto de datos de 86000 archivos aproximadamente el sistema demoraría menos de una hora terminar su procesamiento, comparando un subconjunto de éstos con el total.

Conclusiones y Recomendaciones

Una vez realizadas las pruebas y con los resultados obtenidos, podemos concluir que:

1) La eliminación de palabras vacías en el proceso de transformación de documentos binarios a archivos de texto plano, optimiza la comparación con el algoritmo de Smith-Waterman debido a que se considera una menor cantidad de palabras y por ende matrices de menor tamaño en memoria.
2) La representación de los documentos en archivos de texto plano resulta la más conveniente, debido a que se utiliza una implementación optimizada para su parseo.
3) El algoritmo de Smith-Waterman implementado con las modificaciones propuestas por Robert Irving presentan una mayor exactitud a la hora de establecer secuencias similares (no necesariamente en el mismo orden) entre dos cadenas sujetas a evaluación. Con las pruebas realizadas se pudo observar que aproximadamente representa una mejora en un 155% comparándola con su original.
4) Las modificaciones realizadas al algoritmo en general:
    - Implementación de *stopwords* (palabras vacías).

- Utilización de Hashmaps para el parseo de archivos.
- Utilización de JAMA para la representación de las matrices S y M.

Presentan una mejora en tiempo de procesamiento de aproximadamente un 127% comparándola con su original en el proceso de determinar en qué grado un archivo de texto plano es copia de otro u otros archivos.

5) La implementación de este y otros tipos de algoritmos en un ambiente distribuido no representa mayor demanda en cuanto a cambios en la estructura lógica inicial diseñada para un contexto mono-procesador. Debido a que, en este y en muchos casos, el análisis se realiza de manera no secuencial por archivo.
6) El comportamiento de los nodos, trabajando por demanda de volumen de datos, fue el esperado, presentando una tendencia a la baja, cuando se aumentaba la cantidad de clústeres utilizados con una cantidad constante de datos, que se estabiliza en la utilización de aproximadamente 8 nodos; y presentando una tendencia a la alta cuando el volumen de datos aumenta en un escenario constante de Nodos.

Así mismo, gracias al desarrollo del presente y a las problemáticas con las que nos enfrentamos, podemos recomendar para implementaciones futuras que:

1) En la implantación del módulo de detección de plagio en un Sistema de Administración de Cursos ya operativo, se debe considerar:

- Realizar la transformación de documentos a archivos de texto plano automáticamente cuando un archivo es subido al Sistema de Gestión de Cursos.
- Realizar la copia de este archivo de texto plano al S3 de manera síncrona.
- Registrar el código de este archivo con su ubicación en una tabla de la base de datos utilizadas en el sistema general.

2) El consumo del archivo resultante después del procesamiento debería ser capturado y almacenado en una tabla de la base de datos utilizada en el sistema general, para llevar un histórico del análisis.
3) Dependiendo del volumen de datos que se desee analizar se optaría por una cantidad determinada de nodos para el efecto. Una cantidad óptima de clústeres levantados para un volumen de datos de aproximadamente 65000 archivos (1126 Mb) sería de 15.

Anexos

Trabajo de Ecología y Educación Ambiental

Diego Lavayen Alarcón Paralelo **37**


ARCHIVO / EL UNIVERSO

Las imágenes presentadas en la página www.equilibrioazul.org sin duda alguna pueden motivarme a comentar sobre la vida que poco a poco se está perdiendo gracias a la imprudencia e ignorancia del hombre... Tantas morenas y peces muertos, aletas de tortugas regadas por el suelo marino, tantos tiburones blancos (incapaces de atacar a un humano) muertos solo por el capricho humano, y sólo esto para representar la brutal matanza que día a día se comete no sólo en las costas del Ecuador, sino a nivel mundial.

Todo esto me lleva a pensar en la vida de los Pingüinos de las Galápagos que pronto se verá apagada, quizás n directa del hombre, ya que todavía no le en comercial" a la carne de estos animalitos, sino p del conjunto de ellos, llamado tierra; durante est lo que más ha provocado el hombre, es acelerar el proceso natural de "Calentar tanta emanación de gases tóxicos a la atmósfera, logrando con esto, que Pingüinos Tropicales que existe en el mundo, habitantes de las mundialm Galápagos, vea poco a poco su fin.

Quizás no todos seamos culpables gracias a que tratamos, en la medic aportar en la conservación ambiental, tratando de reciclar y conservar limpio el nosotros; lo que realmente es contraproducente es el "*boom*" que han tenido l últimos 100 años produciendo para sí mismos tanta ganancias, y para el mundo

La realidad es que la población de los Pingüinos de las Galápagos se ha 50% desde la década de los 70', gracias al aumento en 0,8°C a la temperatura se desenvuelven debido al Fenómano de El Niño. Los científicos de la WWF (l la Naturaleza) según sus siglas en inglés, anuncian que si la temperatura se in más, no solo estas aves, sino muchas en el mundo podrían, muy pronto, ll

Repeticiones de El Niño están pronosticadas a consecuencia del cambio está sufriendo el planeta y podrán reducir aún más la ya pequeña población de al borde de la extinción.

Otros de los tantos animales que se encuentran en peligro de extinción s

EL OSO DE ANTEOJOS, maravilloso animal de gran tamaño, que debe su n amarillentas alrededor de sus ojos, como si usara dos grandes lentes, es una d fauna silvestre ecuatoriana que se encuentra asociado a un tipo de hábitat am furtiva, por su carne, su piel y su grasa constituye las principales causas de la poblaciones de estos animales.

MONO CHORONGO O BARRIGUDO, estos preciosos animales, de color castaño, han sufrido la destrucción de su hábitat, afectando seriamente a su

capturados como mascotas por la gracia y encanto que demuestran al ser domesticados. Acostumbran formar numerosos grupos, mismos que se encuentran restringidos a zonas donde no existe cacería ni influencia humana.

TAPIR AMAZONICO O VACA DE MONTE, perseguida por su piel, grasa y uñas. Son criadas en corrales de algunas comunidades en la amazonía, pues su carne es muy deliciosa. La destrucción del bosque primario y secundario, zonas en las que habita, son causas también, del peligro de extinción. A pesar de ello existen personas e instituciones que realizan proyectos por salvar al tapir amazónico, con programas que mantengan su conservación, así podemos encontrar este animal en el Centro Experimental Fátima, ubicado en la ciudad del Puyo, en la amazonía.

EL CONDOR ANDINO, denominado el Rey de los Andes, por su majestuosidad y capacidad de volar hasta los límites de las nieves perpetuas. Considerada el ave voladora más grande del planeta; de coloración negra con contrastes blancos, su hábitat destruido, cazada erróneamente por campesinos y hacendados, se encuentra en serio peligro de extinción, por lo que es imprescindible realizar los más grandes esfuerzos para evitar su extinción.

LOROS, GUACAMAYOS Y PERICOS, sus colores vistosos y llamativos, la capacidad de repetir palabras, sobre todo si son capturadas de jóvenes (loras); la facilidad de adaptarse en cautiverio, los precios altos que por ellas se pagan en el mercado extranjero, la tala indiscriminada de sus bosques y la persecución de la gente local como fuente de proteína, han determinado que estas aves se encuentren restringidas a sitios de protección.

EL PUMA O LEON AMERICANO, es de fomida y esbelta figura, piel leonado rojiza, clara u oscura y hasta medio negra con una mancha del mismo color a cada lado (Patzelt,E1978). Considerada una amenaza para las poblaciones locales y sus animales domésticos, han permitido que desde hace mucho tiempo sean severamente cazadas, además del valor comercial de sus pieles. También la destrucción de sus hábitats han amenazado su conservación (Tirira,D.1999).

PECARI O SAHINOS, en el Ecuador existen dos especies, el pecarí de collar y el pecarí de labio blanco. Estas dos especies se ven amenazadas por la cacería y la destrucción de su hábitat. Su carne es deliciosa y su piel muy cotizada. Se alimentan de frutos, raíces, bulbos, rizomas y a veces de pequeños vertebrados e invertebrados; por lo que se hace necesario conservar ambientes en donde se alimenta estos ejemplares.

LA PACARANA, un roedor de coloración gris con manchas blancas. Considerada como una especie rara en el Ecuador, de movimientos lentos y costumbres nocturnas. Habita en áreas boscosas, se alimenta de frutos y hojas tiernas. Se sienta sobre sus extremidades posteriores para consumir su alimento, mientras que con las extremidades anteriores lo lleva a boca. Es un dispersor de semillas en el bosque. Es una especie muy cazada activamente y debido a su poca distribución, y a la destrucción de su hábitat, esta disminuyendo.

Este tipo de especies se las puede encontrar restringidas en algunos sitios protegidos del Ecuador, en donde turistas nacionales y extranjeros pueden disfrutar de su belleza natural. Estos sitios son: Area de Recreación Cajas, Parque Nacional Sangay, Parque Nacional Podocarpus, Parque Nacional Cotopaxi, Reserva Cayambe Coca, Reserva Cotacachi Cayapas, Parque Nacional Ilinizas, Parque Nacional Antisana, Reserva de Producción Faunística Cuyabeno , Parque Nacional Yasuni, Riveras del rio Napo, Area de Recreación El Boliche, Reserva Biológica Limoncocha, Reserva Ecológica Machi Chindul.

Pero también podemos encontrarlos en sitios en los que se mantienen a la vista del público en general, tales como los Zoológico, San Martin de Baños y Guayllabamba de Quito.

Anexo 1 Documento "Trabajo de Ecología.doc"

Extinción

ECOLOGÍA Y EDUCACIÓN AMBI

Trabajo perteneciente a: Eduardo Cruz Ramírez

Extinción Paralelo **41**

La realidad es que la población de los Pingüinos de las Galápagos se ha reducido a casi un 50% desde la década de los setenta, gracias al aumento en lo casi inadvertido 0,8° C en la temperatura del medio en el que se desenvuelven debido al Fenómeno de El Niño. Los científicos del Fondo Mundial para la Naturaleza, anuncian que si la temperatura se incrementa en 1,2° C más, no sólo estos animales sino muchos en el mundo podrían llegar a la extinción, como:

Extinción Paralelo **41**

EL CONDOR ANDINO, denominado el majestuosidad y capacidad de volar hasta los límite Considerada el ave voladora más grande del planeta; de co blancos, su hábitat destruido, cazada erróneamente por encuentra en serio peligro de extinción, por lo que es in grandes esfuerzos para evitar su extinción.

EL OSO DE ANTEOJOS, maravilloso anima nombre a las franjas amarillentas alrededor de sus ojos lentes, es una de las especies de la fauna silvestre e asociado a un tipo de hábitat amenazado.

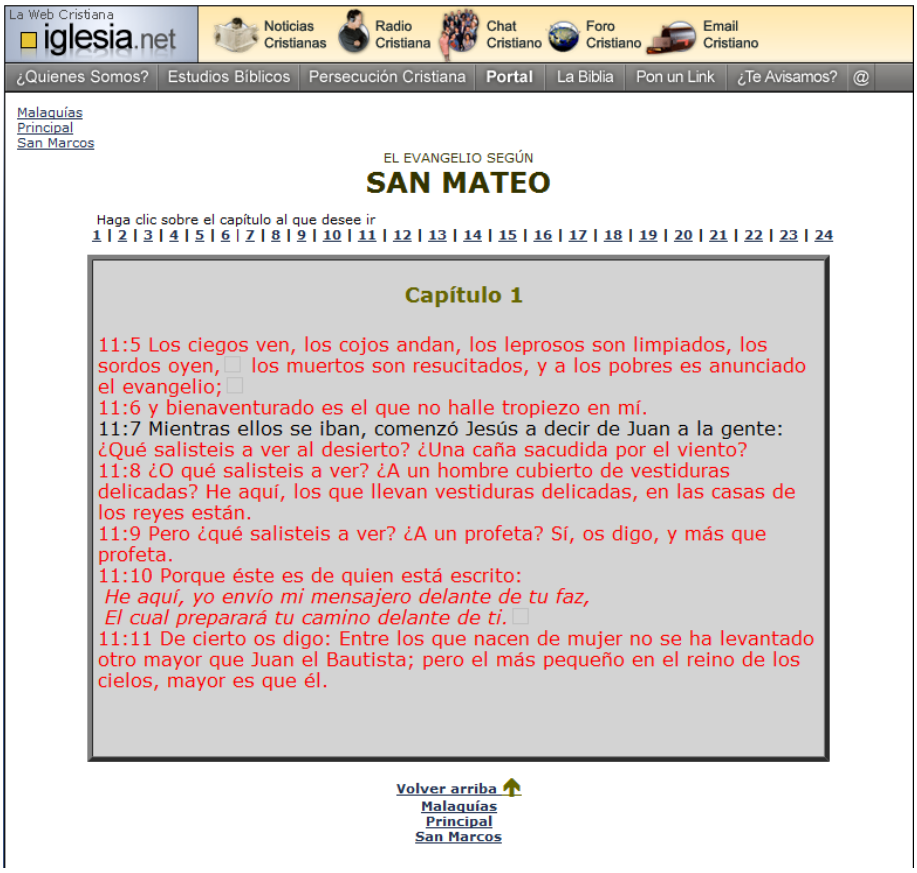
Extinción Paralelo **41**

MONO CHORONGO O BARRIGUDO, de coloración café oscura o castaño, han sufrido la destrucción de su hábitat, afectando seriamente a sus poblaciones. Son capturados como mascotas por la gracia y encanto que demuestran al ser domesticados. Acostumbran formar numerosos grupos, mismos que se encuentran restringidos a zonas donde no existe cacería ni influencia humana.

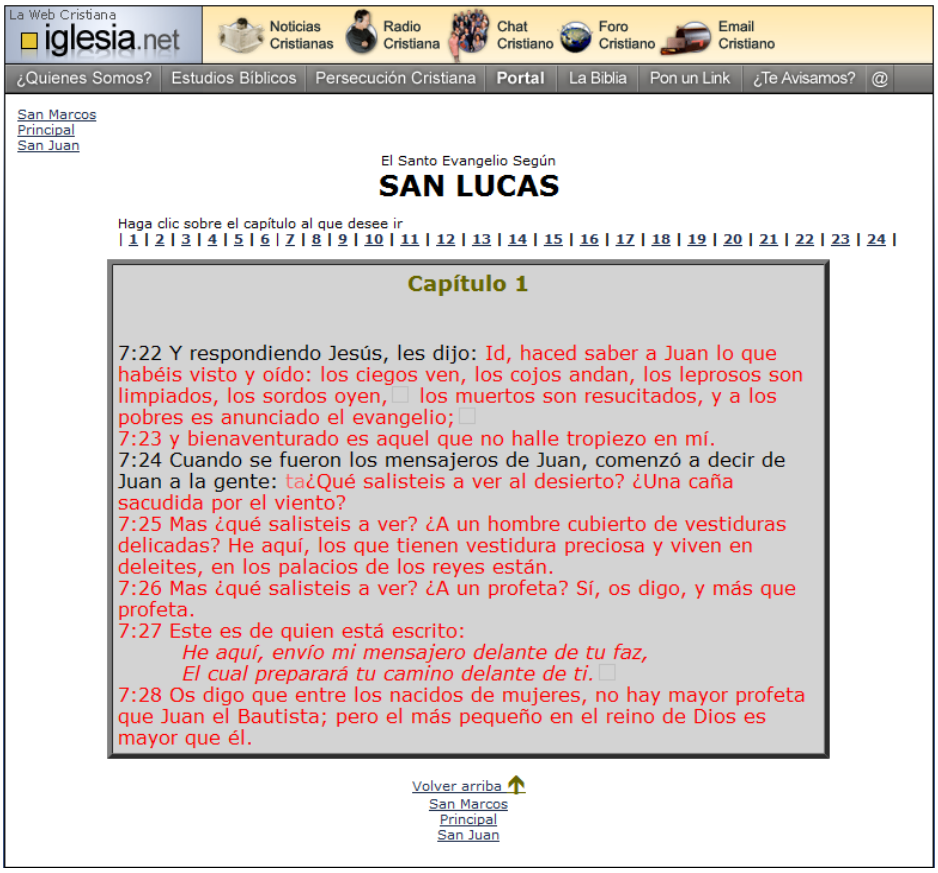
EL PUMA O LEON AMERICANO, es de fornida y esbelta figura, piel leonado rojiza, clara u oscura y hasta medio negra con una mancha del mismo color a cada lado (Patzelt,E1978). Considerada una amenaza para las poblaciones locales y sus animales domésticos, han permitido que desde hace mucho tiempo sean severamente cazadas, además del valor comercial de sus pieles.

LA PACARANA, un roedor de coloración gris con manchas blancas. Considerada como una especie rara en el Ecuador, de movimientos lentos y costumbres nocturnas. Habita en áreas boscosas, se alimenta de frutos y hojas tiernas. Se sienta

Anexo 2 Documento "Deber de Ecología y educación Ambiental.pdf"



Anexo 3 Documento mateo11.5-11.11html – originalmente modificado de http://www.iglesia.net/biblia/libros/mateo.html



Anexo 4 Documento lucas7.22-7.28.html – originalmente modificado de http://www.iglesia.net/biblia/libros/lucas.html

Referencias Bibliográficas

[1] S.F. Altschul, W. Gish, W. Miller, E.W. Myers, and D.J. Lipman. “A basic local aligment search tool”. Journal of Molecular Biology, 215:403 – 410. Octubre 1990.

[2] Smith TF, Waterman MS. “Identification of common molecular subsequences”. J Mol Biol. 147 (1): pp. 195-7. PMID 7265238. Marzo 1981.

[3] Robert W Irving. “Plagiarism and Collusion Detection using the Smith-Waterman Algorithm”, University of Glasgow. Julio 2004.

[4] Apache. Hadoop. “http://hadoop.apache.org/”. Febrero 2010

[5] White, Tom. “Hadoop: The Definitive Guide”. O'Reilly Media. pp. 524. ISBN 0596521979. Mayo 2009.

[6] T. Lancaster. “Effective and Efficient Plagiarism Detection”. PhD thesis, School of Computing, Information Systems and Mathematics, South Bank University. Enero 2003.

[7] X. Huang, R.C. Hardison, and W. Miller. “A space-efficient algorithm for local similarities”. CABIOS, 6:373 – 381. Marzo 1990.

[8] Z. Zhang, P. Berman, and W. Miller. “Alignments without low scoring Regions”. Journal of Computational Biology, 5:197 – 210. Febrero 1998.