udp FACULTAD DE INGENIERÍA

Facultad de Ingeniería
Escuela Ingeniería Informática

Compresión de datos distribuida basado en clustering

Guillermo Andrés Navarro Giglio

Memoria para optar al título de Ingeniero Civil en Informática y Telecomunicaciones

Profesor guía
Francisco Claude-Faust

Comité
Roberto Konow

Julio, 2015

UNIVERSIDAD DIEGO PORTALES
Facultad de Ingeniería
Escuela Ingeniería Informática

Compresión de datos distribuida basado en clustering

Guillermo Andrés Navarro Giglio

Memoria para optar al título de Ingeniero Civil en Informática y Telecomunicaciones

---

Francisco Claude-Faust
Profesor guía

Roberto Konow
Comité

Julio, 2015

Dedico esta memoria a todas las personas.

Contenido

Lista de figuras

Lista de tablas

Resumen

Hoy en día la información crece rápidamente con nuevos contenidos provenientes de páginas Web, redes sociales, aplicaciones móviles entre otros. Cada vez es más difícil poder manejar esta gran cantidad de datos, demandando mayores recursos para las empresas y transformándose en un importante desafío en el futuro. Para esto se debe buscar nuevos mecanismos que permitan de manera eficiente almacenar esta gran cantidad de datos.

Está memoria intenta buscar un mecanismo eficiente de distribuir una colección de datos para luego obtener el máximo de compresibilidad manteniendo un balance en lo que respecta al uso de recursos en multiples servidores. Primero se explica la importancia de los documentos versionados en la actualidad y por que se hace necesario distribuir los datos, luego se implementa un mecanismo de distribución de la colección de datos y finalmente se forma una conclusión de los resultados obtenidos. Esto último nos permite plantear nuestro programa de avance para la segunda etapa de esta memoria.

Capítulo 1

Introducción

1.1. Antecedentes generales

¿La información tiene límites? ¿Somos capaces de guardar toda esta información? Hoy en día la información crece a pasos agigantados; cada día aparecen nuevos contenidos provenientes de páginas Web, redes sociales, aplicaciones móviles y de nuevas tecnologías como Internet de las cosas, que son capaces de generar una gran cantidad de información.

Además, esta información puede ir cambiando con el pasar del tiempo y en algunos casos, es necesario ser capaz de guardar el historial de cambios de esta. Ejemplos de aplicaciones que tienen estos requerimientos son por ejemplo: Wikipedia, una enciclopedia online colaborativa [1], y Git, un manejador de versiones utilizado principalmente para almacenar código de fuente [2]. Herramientas tradicionales de almacenamiento y versionamiento no son capaces de manejar una gran cantidad de datos de manera eficiente o incluso práctica que generan mayores costos, transformádonse en un verdadero desafío para las empresas.

Frente a este escenario, almacenar los datos de una aplicación masiva es cada vez menos viable usando un solo computador. Empiezan a aparecer nuevas soluciones a este problema, por ejemplo, la empresa Backblaze, desarrolla una aplicación que genera una copia de seguridad en la nube a muy bajo costo, usa una granja de servidores para almacenar los datos, repartiendo la información entre un conjunto de computadores que forman un sistema distribuido [3]. Se deben buscar nuevos mecanismos que logren de manera eficiente almacenar los datos aprovechando los recursos, que se traduce en una disminución en los costos de las empresas.

Esta memoria consiste en estudiar una alternativa basada en clustering para repartir la información de manera inteligente entre varios computadores, haciendo uso de una métrica de similitud basada en el contenido de la información para luego comprimirla, y así de esta forma, se espera mejorar la compresión, idealmente manteniendo un balance en la carga de almacenamiento [4]. Clustering es una técnica que genera agrupaciones de objetos según un criterio, se pretende con esto separar los datos de tal manera que en cada agrupación se tenga un espacio parecido con respecto a los demás y que al utilizar cierto compresor sea mucho más eficiente en término de espacio que solamente separar los datos de manera aleatoria, logrando un ahorro en espacio en disco mucho mayor.

Definiciones:

String o cadena de texto: Es cualquier cadena de símbolos ya sea binario o texto en cualquier formato de codificación de caracteres.
Coleccion de documentos: Conjunto de String.
Compresión: Es la reducción del volumen en datos, comprimir tiene un límite definido por la entropía (informacion nueva que se define como la cantidad total de datos menos su redundancia).
Puntos: Se define como un vector de n dimensiones, $P =< v_1, v_2, ..., v_n >$.
Distancia de similitud: Se define como la distancia entre dos puntos a partir de la similitud de estos.
Agrupamiento: Se define como el acto de formar grupos de puntos que sean en lo mayor posible similares entre si y poco similares con respecto a puntos de otros grupos.
Wiki-ES: Wikipedia en español.
Wiki-EN: Wikipedia en inglés.

1.2. Objetivos

El objetivo general de esta memoria es maximizar la compresibilidad de la colección agrupándolos en función de su similitud de contenido, para esto se quiere implementar y evaluar un mecanismo que represente colecciones versionadas de datos de forma distribuida.

1.2.1. Objetivo específicos

- Realizar un estudio de mecanismos de clustering para agrupar conjuntos de Strings.
- Generar un repositorio de datos distribuido basado en clustering.
- Medición de la efectividad de usar clustering previo a la compresión para el almacenamiento distribuido de datos, en términos de la compresibilidad de la colección.

1.3. Plan de trabajo

En la tabla 1.1 se presenta el plan de trabajo. Este plan tiene algunas alteraciones mínimas en las fechas y tareas con respecto al plan original.

Tarea	Fecha inicio	Fecha final	Entregables
Anteproyecto	20-03-2015	08-04-2014	Anteproyecto
Estudiar y probar agrupamientos	08-04-2015	30-04-2015	Resultados experimentales en un set de datos preliminar
Implementar un separador de datos basado en agrupamientos	30-04-2015	10-05-2015	Sistema de agrupamiento funcionando
Documentación	10-05-2015	25-07-2015	Tesis I
Mejorar algoritmo de agrupamiento	25-07-2015	17-09-2015	Algoritmo de agrupamiento
Realizar pruebas y comparaciones	17-09-2015	15-10-2015	Análisis de Resultados
Documentación final	15-10-2015	01-12-2015	Tesis II

Tabla 1.1: Plan de trabajo.

1.4. Metodología

La metodología de desarrollo de la memoria será con reuniones semanales con el profesor guía revisando los avances y pasos a seguir, se contará con un repositorio Git que contendrá el código fuente desarrollada y las fuentes de la memoria misma.

En una primera etapa se estudiara sobre los distintos mecanismos de agrupamiento, luego se analizara y diseñara un algoritmo de agrupamiento para string. Después se implementara el algoritmo de agrupamiento diseñado en una computadora personal para realizar pruebas pequeñas y comprobar su correcto funcionamiento, si las pruebas iniciales son satisfactorias se realizan las pruebas con una colección de datos mayor, en el servidor. Luego se evalúan los resultados obtenidos, en base a los resultados se evalúa como mejorar el algoritmo implementado y se vuelve a repetir el proceso de implementación del algoritmo para ir mejorando los resultados.

Capítulo 2

Conceptos Básicos

2.1. Algoritmo de agrupamiento

El algoritmo de agrupamiento es un proceso en la cual se tiene un conjunto de puntos y se crean grupos de puntos a partir de una medida de similitud, en la mayoría de los algoritmos de agrupamiento se asume un espacio euclidiano .

El espacio euclidiano es un espacio geométrico que se cumplen los axiomas de Euclides, se deben cumplir tres reglas entre las distancias de dos puntos en el espacio euclidiano:

- La distancia entre los puntos nunca es negativo y solamente es 0 consigo mismo.
- La distancia es simetrica, es decir, no importa si se calcula la distancia del punto x a y o de y a x.
- La distancia obedece a la desigualdad del triangulos; la distancia de x a y a z no puede menor a la distancia de z a x.

Los algoritmos de agrupamiento se pueden dividir en dos grupos, jerárquicos y no jerárquicos. En los algoritmos de agrupamiento jerárquicos pueden ser aglomerativos o divisivos.

Cuando es aglomerativo, cada punto en el espacio euclidiano representa un grupo y en cada iteración se unen los grupos hasta llegar a los números de grupos deseados, en cambio, los divisivos todos los puntos se encuentran en un sólo grupo y en cada iteración se dividen los grupos.

Para entender mejor el concepto del algoritmo de agrupamiento a continuación se explica *k-means* [5], que es unos de los algoritmo más utilizado y simple que se utiliza hoy en día.

K-means como la mayoría de los algoritmos de agrupamiento asume un espacio euclidiano y también asume el número de agrupos conocidos. El número de grupos se puede determinar de distintas maneras, como por ejemplo, por prueba y error o con conocimiento previo de las características de las observaciones.

Dado k grupos se genera k centros de grupos o centroides iniciales,los centroides son vectores de las medias de las características de todas las observaciones dentro de cada grupo. Los centroides se pueden asignar de distintas maneras, una opción es asignar observaciones aleatoriamente de un conjunto de observaciones. Luego se itera los siguientes pasos:

Algoritmo 1 Algoritmo K-means

```
C conjunto de k centroides;
while true do
  Para cada observacion calcular la distancia a todos los C_i;        #
  0 < i < k

  Las observaciones se asignan al C_i con la media más cercana;
  C_i se recalculan las medias con las nuevas observaciones agregadas;
  if Todos los C_i no cambian then
    Termina
  end if
end while
```

Se itera hasta que converga, es decir, ya no se asignan nuevas observaciones a los grupos o los centroides ya no se mueven.

La mayoría de los algoritmos de agrupamiento asume un espacio euclidiano para medir la similitud entre los vectores, pero en el caso de las cadenas de texto no son puntos que se puedan representar en el espacio euclidiano.

Para poder solucionar este problema se debe buscar un mecanismo para medir la similitud entre las cadenas de texto. Existen varias medidas de similitud para determinar la similitud entre cadenas de texto, en la siguiente sección se nombran algunas de las más utilizadas junto con la medida que se utilizó en la memoria la implementación del algoritmo de agrupamiento.

2.1.1. Medida de similiud para Cadenas de texto

Unos de los puntos más importante al momento de implementar un algoritmo de agrupamiento es la medida de similitud entre los puntos, en este caso los puntos representan las cadenas de texto y se debe buscar una medida de distancia.
Existen varios métodos para calcular la similitud entre un string y otro como por ejemplo:

- Distancia de Edición: Se tiene dos cadenas de texto A y B.La distancia de edición es la cantidad mínima de insertar, sustituir o eliminar necesarios para transformar A en B. Con distancia de edición se logra una buena calidad en la similitud, pero presenta una desventaja: el algoritmo de edición de distancia requiere de tiempo de $O(n \times m)$, donde n y m son el largo de ambas secuencias de strings. Por ejemplo para las cadenas "abracadabra" y "alabaralabarda", se necesitan 7 operaciones.

- Jaccard: Es una medida de similitud que está definido por el tamaño de la intersección de dos secuencias divido por el tamaño de la unión de ambas secuencias, un ejemplo en la medida de similitud entre las secuencias "night" y "nacht" es de 0.3. Se tiene el conjunto S y T la formula para determinar el similitud jaccard es 2.1

$$Jaccard = \frac{(S \cap T)}{(S \cap T)}, \tag{2.1}$$

- Distancia Hamming: Se tiene dos cadenas de caracteres de igual tamaño y se calcula cantidad de sustituciones necesarios para transformar una cadena de caracteres en otra. Por ejemplo "night" y "nacht" se necesita 2 sustituciones.

En esta memoria se quiere una medida de distancia basada en compresión, la medida de similitud implementada para el algoritmo de agrupación es una distancia de compresión utilizando algún método de compresión como lzma, gzip o bzip [6]. Se aplica la siguiente fórmula para obtener la *Similitud* 2.2

$$Similitud = \frac{(d_{1+2} - d_2)}{d_1}, \tag{2.2}$$

donde d_1 es el tamaño comprimido del documento más grande,d_2 es el tamaño comprimido del documento más pequeño, y d_{1+2} es el tamaño comprimido de la unión de d_1 y d_2 sin comprimir.

El valor de la variable d_{1+2} va a depende de la similitud de los strings comprimidos, si los strings tienen un grado de similitud la compresión será mucho más efectiva ya que se necesita un diccionario mucho menor para comprimir, al contrario ocurre cuando los strings son muy distintos entre sí. Entre más pequeño el valor de la variable *Similitud*, significa que ambos documentos son muy similares, y entre más grande los valores significa que los documentos son diferentes. En comparación con la distancia edición está obtiene una buena calidad de similitud, pero el tiempo de ejecución es lineal. Mejorando el tiempo $O(n \times m)$ de la distancia de edición. Esto hace una buena opción al momento de seleccionar una medida de similitud para grandes colecciones de datos. Esta medida de similitud como entrega solamente un valor es de una sola dimensión.

Por ejemplo, si se utiliza el compresor LZ78 [7] en la secuencia S_1 = *'abracadabra'*, S_2 = *'abracadadah'* y la suma S_{1+2}=*'abracadabraabracadadah'*, al aplicar la formula 2.2 se obtiene el valor 0,71.

Ahora si cambiamos la segunda secuencia a S_3= 'casasyperro', la cual no tiene similitud con la primera secuencia, se obtiene el valor 0.77. De esta forma se observa que con menor similitud entre los strings, mayor es el valor, entonces al tener strings que son similares el valor se acerca al 0 .

Capítulo 3

Algoritmo de agrupamiento propuesto

El algoritmo de agrupación implementado para la distribución de las cadenas de texto es una variante del algoritmo cure [5] que utiliza un algoritmo jerárquicos para formar los grupos iniciales.

A continuación se muestra las etapas que sigue el algoritmo implementado:

1. Primero se selecciona una muestra de la coleccion de datos y se aplica un algoritmo jerarquico.

2. Luego de formar los grupos iniciales se asigna cada string de la colección al grupo más cercano.

3. Finalmente se comprime cada agrupación.

En el primer paso se debe seleccionar un algoritmo jerárquico, la implementación es un algoritmo aglomerativo. Se ingresa cada cadena de texto de la muestra a un grupo, y se calcula la similitud de las cadenas de texto según la medida de distancia nombrada anteriormente **??**, después se selecciona ambas cadenas de texto que tengan la menor distancia y se unen los grupos de cada cadena de texto, antes de unirlos se pueden agregar varias reglas, por ejemplo, que las agrupaciones tengan un límite en el número de cadenas de texto, esta medida puede servir para balancear los grupos.En la segunda parte del algoritmo, a diferencia de cure que selecciona del grupo n puntos más alejados para representar al grupo, llamados *puntos representativos*,el algoritmo base toma todos los puntos como representativos. Unos de los objetivos que se busca es obtener un balance del espacio ocupado en cada agrupación, para esto la cadena de texto antes de ser ingresada al grupo más cercano y se tiene k agrupaciones, se comprueba que el tamaño en espacio de memoria no supera la $k - decima$ parte del espacio en memoria del total de la colección de cadenas de texto, para mayor claridad se presenta la siguiente formula 3

$$Grupo_i < \frac{C}{k}, \tag{3.1}$$

donde $Grupo_i$ es el tamaño en disco del grupo más cercano a la cadena de texto seleccionada, C es el tamaño en disco de la Colección de cadenas de texto, k es el número de grupos.
Con esta comprobación al momento de asignar una cadena de texto a un grupo se busca generar un balance en cada grupo, si el $Grupo_i$ de una agrupación es mayor entonces se comprueba el siguiente grupo más cercano a la cadena de texto seleccionada hasta encontrar un grupo que sea menor. Si bien se genera un balance en el espacio de memoria la comprobación se realiza antes de la compresión y puede ocurrir que al momento de la compresión no asegura un balance notorio en todas las agrupaciones.
El Algoritmo 2 muestra la función similitud que representa el resultado de la medida de distancia entre dos cadenas de texto.
El algoritmo 3 muestra el pseudocodigo del algoritmo de agrupamiento propuesto en la memoria.

Algoritmo 2 Funcion SIMILITUD

Require: $String1$
Require: $String2$
1: $s1 \leftarrow COMPRESS(String1)$
2: $s2 \leftarrow COMPRESS(String2)$
3: $s12 \leftarrow COMPRESS(String1 + String2)$
4: **return** $size(s12) - size(s2)/size(s1)$

Algoritmo 3 Algoritmo de agrupamiento propuesto

Require: $Sampling = \{d_1, \ldots, d_n\}$ #
Muestra obtenida de la colección, donde n es la cantidad de cadenas de texto.
Require: $Collection = \{d_1, \ldots, d_k\}$ #
Colección de datos, donde k es numero de cadenas de texto.
Require: C #
Número de grupos deseados.

1: $S \leftarrow \langle\ \rangle$
2: **for each** s1 in $Sampling$ **do**
3: **for each** s2 in $Sampling$ **do**
4: $S \cup (SIMILITUD(Sampling[s1], Sampling[s2]), s1, s2)$
5: **end for**
6: **end for**
7: Sort(S)
8: $i = 0$
9: **while** $Sampling > C$ **do**
10: $Grupo1 \leftarrow Sampling[S[i][1]]$
11: $Grupo2 \leftarrow Sampling[S[i][2]]$
12: $Grupo1 \cup Grupo2$
13: $i = i + 1$
14: **end while**
15: $Grupo \leftarrow \langle\ \rangle$ #
Grupo lista de tamaño C

16: **for each** d in $Collection$ **do**
17: $i = 0$
18: **for each** $s = 0$ to n **do**
19: **if** $SIMILITUD(Sampling[s], d) > i$ **then**
20: $i = SIMILITUD(Sampling[s], d)$
21: **end if**
22: **end for**
23: $Grupo[i] \cup s$
24: **end for**

3.0.2. Pruebas Iniciales

Las pruebas se realizarán comparando dos tipos de algoritmos de agrupamiento:

- Algoritmo de agrupamiento propuesto
- Algoritmo de agrupamiento random

El Algoritmo de agrupamiento random crea n grupos y asigna de manera uniforme y distribuido aleatoriamente cada cadena de texto a un grupo, a diferencia del algoritmo propuesto anteriormente que es determinista, es decir, que los resultados siempre serán iguales. También mantiene el balance de espacio en memoria en cada Grupo.
Con ambos algoritmos se pretende demostrar que agrupando los grupos de manera inteligente se pueda obtener mejores resultados que agrupándolos aleatoriamente y mantener cierto balance en cada agrupación.
En el algoritmo de agrupamiento propuesto existen distintas variables que pueden determinar un buen agrupamiento de la colección tales como:

- Cantidad de Grupos: Es difícil determinar la cantidad exacta de grupos que se necesita, en nuestro caso es el número de máquinas.
- Tamaño de la muestra: si la muestra es muy pequeña, es muy probable que no represente todos los tipos de grupos que se encuentra en la colección.
- Medida de similitud: Si la medida de similitud no obtiene una buena calidad es posible que agregue puntos de otro grupo, en la compresión se permite determinar el nivel de compresión, mientras que sea mayor el nivel la compresión es mayor pero es mucho más lento.

Las pruebas se realizaron de un dataset que se descargó la primera parte de la versión en inglés de Wikipedia, las muestras se tomaron de manera uniforme al azar. Para cada documento seleccionado se obtiene todas sus versiones. En total el dataset contiene 16384 documentos de 1 MB c/u, en su totalidad es de 16.384 GB. Esto fue hecho usando la libreria go-wikiparse.

Capítulo 4

Resultados

En las pruebas se tomó una muestra no superior a 30 documentos ya que al aumentar la muestra el tiempo de ejecución crece $\frac{n^3}{2}$, donde n es la cantidad de documentos. La muestra se obtuvo aleatoriamente de la colección de documentos, como la muestra es insignificante en comparación al tamaño de la muestra es muy probable que ningún documentos perteneciera a un documento de la misma versión. Este problema origina que la mayoría de los documentos pertenezca a un solo grupo y el resto solamente es representado por 1 documento. También cabe mencionar que la elección de la cantidad de agrupaciones es arbitraria, pero la cantidad de agrupaciones es una variable importante al momento de obtener buenos resultados en las agrupaciones, en este caso las pruebas se realizaron con un número fijo de agrupaciones para observar el comportamientos de otras variables que afectan a las agrupaciones.

En la tabla 4.1 muestra los resultados de cada método con una cantidad de 10 agrupaciones, en cada agrupación se muestra el resultado de la compresión. El *Método 1* utiliza el algoritmo de agrupación aleatoria, se tiene que en cada grupo se mantiene una carga en memoria balanceada que es uno de los objetivos deseados en la memoria. En los métodos siguientes se utiliza el algoritmo de agrupación propuesto pero modificando algunas variables para observar su comportamiento.

Para el caso del *Método 2* se observa una mejora de la compresión equivalente al 45 % del tamaño total del resultado en el algoritmo de agrupamiento aleatorio, aquí la muestra es de 30 documentos. En términos de balance en la carga de memoria, este método es ineficiente ya que la mayor parte de la carga se concentra solamente en un grupo. Esto se debe a que en el momento de crear los grupos con las muestras, la mayor parte de las muestra quedan solamente en un grupo dejando a las demás con pocas muestras de representación.

En el *Método 3* puede apreciar que existe un balance en la cantidad de muestras en cada agrupación. Para esto, cada grupo no tendrá una muestra de documentos superior a 3 en un universo de 30 documentos. Con esto se busca balancear la cantidad de documentos en cada agrupación. El resultado de la compresión utilizando el *Método 3* es equivalente al 55 % del tamaño total del resultado con el metodo del algoritmo de agrupamiento aleatorio, que sigue siendo una mejor alternativa, pero comparando con los resultado del *Método 2* se paga un costo al balancear las muestras en los grupos, de un 24 % más del tamaño total del resultado en el *Método 2*.

En el *Método 4*, se hace la misma prueba que en el método anterior pero se agrega una condición de que el tamaño en espacio de memoria no supere un límite. El límite en este caso es el tamaño en memoria de la colección de datos divido por la cantidad de agrupaciones, con esta medida se asegura que en todos los grupos tienen aproximadamente la misma cantidad de cadenas de texto. El resultado del *Método 4* es el equivalente al 70 % del tamaño total del resultado en el algoritmo de agrupamiento aleatorio que sigue siendo una mejora, pero nuevamente pagando un costo, con respecto al *Método 2* aumenta 55 % más de tamaño, incluso mayor que en el *Método 3*, pero con mejores resultados en el balance de la carga en memoria.

Grupos	Método 1(KiB)	Método 2(KiB)	Método 3(KiB)	Método 4(KiB)
Total	65.510	29.371 (45 %)	36.036 (55 %)	45.639 (70 %)

Tabla 4.1: Resultado algoritmo de agrupamiento aleatorio con 10 Grupos.

En la tabla 4.2 se observa el resultado de la distribucion de los datos de cada grupo utilizando los metodos mencionados.

Grupos	Nº Documentos	Tamaño(KiB)	Distancias	
Grupo 1	6.530	1.089	5.595	5.168
Grupo 2	6.648	1.442	7.102	8.271
Grupo 3	6.491	1.020	3.406	3.831
Grupo 4	6.756	2.851	3.465	3.353
Grupo 5	6.540	857	4.523	3.570
Grupo 6	6.456	1.455	2.729	4.343
Grupo 7	6.562	756	2.678	4.349
Grupo 8	6.486	1.106	2.748	3.239
Grupo 9	6.471	16.551	2.473	4.306
Grupo 10	6.596	2.245	1.318	5.209

Tabla 4.2: Distribución grupos.

Capítulo 5

Trabajo futuro

Se puede concluir que agrupando las secuencias de strings de manera inteligente se obtiene mejores resultados que agrupándolos aleatoriamente, pero intentar balancear la carga de espacio en cada agrupación se paga un costo al comprimir. Parte importante para obtener una buena agrupación es la medida de similitud, para agrupar grandes cantidades de string es necesario que la medida de similitud entre dos string sea rápido ya que cada secuencia de string debe compararse con cada secuencia del sampling. La ventaja de este algoritmo son que los resultados son determinista, es decir, cuantas veces se ejecuta el algoritmo para una misma colección siempre entrega el mismo resultado, entonces al momento de obtener las agrupaciones con el sampling es posible asignar strings en varios procesos rebajando el tiempo de ejecución.

El plan de trabajo para la segunda etapa de la tesis consistirá en intentar mejorar los resultados de la compresión de la colección de datos obtenidos en esta primera parte, por ejemplo, cambiando las distintas variables que influyen en agrupación de strings como el numero de agrupaciones, para obtener mejores resultados en la compresión de las agrupaciones.

Referencias bibliográficas

[1] Wikipedia. https://en.wikipedia.org/.

[2] Git. https://git-scm.com/.

[3] Backblaze. https://www.backblaze.com/blog/storage-pod-4-5-tweaking-a-proven-design/.

[4] David Salomon abd Giovanni Motta. *Handbook of Data Compression*. Springer London, 2012.

[5] Anand Rajaraman and Jeffrey David Ullman. Mining of massive datasets. pages 221–260, 2011.

[6] S. Kreft and G. Navarro. On compressing and indexing repetitive sequences,theoretical computer science. 483:115–133, 2011.

[7] Francisco Claude and Gonzalo Navarro. Improved grammar-based compressed indexes. 12:180–192, 2012.

udp FACULTAD DE INGENIERÍA

Facultad de Ingeniería
Escuela Ingeniería Informática

Compresión de datos distribuida basado en clustering

Guillermo Andrés Navarro Giglio

ANEXOS

Profesor guía
Francisco Claude-Faust

Comité
Roberto Konow

Julio, 2015

Anexo A

CODIGO ALGORITMO DE AGRUPAMIENTO BASE

```
#!/usr/bin/python

""" argv[1]= carpeta de documetnos para generar clusters
        argv[2]= Numero de clusters deseado
        argv[3]= carpeta guardar clusters
        argv[4]= carpeta de documentos
        argv[5]= max cluster size
        argv[6]= calidad distancia documento 0-9
"""
import os, sys ,getopt
import editdist
import shutil
import distance
import zlib


def get_size(start_path = '.'):
    total_size = 0
    for dirpath, dirnames, filenames in os.walk(start_path):
        for f in filenames:
            fp = os.path.join(dirpath, f)
            total_size += os.path.getsize(fp)
    return total_size

def comprimir( ):
        for i in os.listdir(str(sys.argv[3])):
                os.system('7z a '+i+'.7z '+str(sys.argv[3])+i+'/')

def distancia_zip(s1, s2, nivel=6):
        compressed1 = zlib.compress(s1,nivel)
        compressed2 = zlib.compress(s2,nivel)
        compressed12 = zlib.compress(s1+s2,nivel)
        #c1=float(len(compressed1))
        #c2=float(len(compressed2))
        #c12=float(len(compressed12))
```

```
        #return float(c12/(c2+c1))
        if len(compressed1) > len(compressed2):
                n = (len(compressed12) - len(compressed2)) / float(len(compressed1
                    ))
        else:
                n = (len(compressed12) - len (compressed1)) / float(len(
                    compressed2))
        return n



if __name__ == "__main__":

        #numero de clusters
        clusters = int(sys.argv[2])
        #doculmentos para generar los clusters
        data=[]
        #guardar bloque de datos
        for document in os.listdir(str(sys.argv[1])):
                f = open(str(sys.argv[1])+str(document))
                data.append([f.read()])
                f.close()


        #lista documentos ordenados por sus distancias
        test=[]
        print "sampling cargado, buscando las distancias minimas..."
        for k in range(len(data)):
                        for j in range(k+1,len(data)):
                                comp=distancia_zip(data[k][0],data[j][0],int(sys.
                                    argv[6]))
                                print "t1:"+str(k)+" contra t2:"+str(j)+"="+str(
                                    comp)
                                test.append([comp,data[k][0],data[j][0]])
        test.sort()

        print len(test)


        print "creando clusters"
        i=0
        while len(data)>1 and len(data) > clusters and i < len(test):
                cluster1=[]
                cluster2=[]
                t1=False
                t2=False
                for j in data:
                        if t1 == True and t2 == True :
                                break
                        for k in range(len(j)):
                                if test[i][1] == j[k]:
                                        t1=True
                                        cluster1=data.index(j)
                                if test[i][2] == j[k]:
                                        t2= True
                                        cluster2=data.index(j)
                i=i+1
                if cluster2 ==cluster1 or (len(data[cluster1])+len(data[cluster2])
                    ) > int(sys.argv[5]) :
                        continue
                test3=data.pop(cluster2)
                if cluster2<cluster1:
                        cluster1-=1

                test4=data.pop(cluster1)
                data.append(test3+test4)

                densidad=[]
                print "densididad sampling clster:"+str(test[i][0])
                for t in data:
                        densidad.append(len(t))
                print densidad

        densidad=[]
```

```
print "IMPRIMIR CLUSTER"
for t in data:
        densidad.append(len(t))
print densidad

print "Creando carpetas de clusters..."

clust_size=[]
for t in range(len(data)):
        os.mkdir( sys.argv[3]+str(t));
        clust_size.append(0)

div=get_size(str(sys.argv[4]))/len(data)

i=0

print "Asignando documentos a los clusters"
for document in os.listdir(str(sys.argv[4])):
        test2=[]
        f = open(str(sys.argv[4])+str(document))
        texto=f.read()
        doc_z = len(texto)
        for j in data:
                for k in range(len(j)):
                        #comp=editdist.distance(j[k],texto)
                        #comp=distance.hamming(j[k],texto)
                        comp=distancia_zip(j[k],texto,int(sys.argv[6]))
                        test2.append([comp,data.index(j)])
        test2.sort()

        for s in test2:
                if clust_size[s[1]] < div:
                        shutil.copyfile(str(sys.argv[4])+str(document),
                             sys.argv[3]+str(s[1])+'/'+str(document))
                        clust_size[s[1]]+=doc_z
                        break
        f.close()

for s in clust_size:
        print s

comprimir()
```

Anexo B

CODIGO ALGORITMO DE AGRUPAMIENTO ALEATORIO

```
#!/usr/bin/python
import glob
import os,sys
import random
import shutil

""" argv[1]= carpeta de documentos
        argv[2]= n clusters
        argv[3]= carpeta resultados
"""

print sys.argv[1]
lista=os.listdir(str(sys.argv[1]))
random.seed()
print len(lista)


print "Creando carpetas de clusters..."

for t in range(int(sys.argv[2])):
        os.mkdir( sys.argv[3]+str(t));

div=len(lista)/int(sys.argv[2])
print div



for i in range(int(sys.argv[2])) :
        for j in range(div) :
                rand = random.randint(0,len(lista)-1)
                texto= lista.pop(rand)
                shutil.copyfile( sys.argv[1]+texto , sys.argv[3]+str(i)+"/"+texto)
```