udp FACULTAD DE INGENIERÍA

Facultad de Ingeniería
Escuela Ingeniería Informática

Compresión de datos distribuida basado en clustering

Guillermo Andrés Navarro Giglio

Tesis 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

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

---

Francisco Claude-Faust
Profesor guía

Roberto Konow
Comité

(Julio), (2015)

Dedico esta tesis a todas las personas que me apoyaron.

Contenido

Lista de figuras

Lista de tablas

Abstract

Keep versioned documents it is important when you recall information prior to the current version but have a great cost in memory space, making it necessary to distribute this data. This thesis proposes to implement a mechanism for grouping of data according to their similarity for maximum compressibility possible and ideally the groups have similar sizes to compress.

Resumen

En esta tesis intenta buscar un mecanismo eficiente de distribuir una colección de datos para luego obtener el máximo de compresibilidad. 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.

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[5], y git, un manejador de versiones utilizado principalmente para almacenar código de fuente[6]. Herramientas tradicionales de almacenamiento y versionamiento no son capaces de manejar una gran cantidad de datos.

Frente a este escenario, almacenar los datos de una aplicacion masiva es cada vez menos viable usando un solo computador. 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[2].

Esta tesis 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 [3]. Clustering es un algoritmo 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.

Capítulo 2

Objetivos

2.1. Objetivos

El objetivo general de esta memoria es implementar y evaluar un mecanismo que represente colecciones versionadas de datos de forma distribuida y que estos se encuentren agrupados en función de su similitud de contenido de forma de maximizar la compresibilidad de la colección.
Las colecciones que se utilizarán en la memoria son Wiki-ES, Wiki-EN y el Kernel de Linux.

2.2. Objetivo específicos

- Realizar un estudio de mecanismos de clustering para agrupar conjuntos de Strings.
- Generar la implementación de 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 compresibilidad de la colección.

Donde String es cualquier cadena de símbolos ya sea texto o binario.

Capítulo 3

Solución

3.1. Algoritmo de agrupamiento

El algoritmo de agrupamiento es un proceso en la cual se tiene varios "puntos"suponiendo que se encuentra en un espacio eucladiano y se crean grupos de esos "puntos.[en] funcion de una medida de similitud.

Los algoritmo de agrupamiento se pueden dividir en dos grupos , jerarquicos y no jerarquicos. En los algoritmo de agrupamiento jerarquicos pueden ser aglomerativo o divisivo. Cuando es aglomerativo cada punto en el espacio eucladiano representa un cluster y en cada iteracion se unen los cluster hasta llegar a los numeros de clusters deseados, en cambio, los divisivos todos los puntos se encuentran en un solo cluster y en cada iteracion se divide los clusters.

3.1.1. Medida de distancia

Unos de los puntos mas importante al momento de implementar un algoritmo de agrupamiento es la medida de similitud o de distancia entre los "puntos", en este caso los "puntosrepresentan los strings.

Existen varios metodos para calcular la similitud entre string como por ejemplo : coseno.

- Edicion de distancia : Se tiene dos cadenas de caracteres A y B Es la cantidad minima de insertar,sustitución o eliminar necesarios para transformar de A en B. Con edicion de distancia se logran una buena calidad en la similitud, pero presenta una desventaja, el algoritmo de edicion de distancia es de $O(n * m)$, donde n y m son el largo de ambas secuencias de strings. Por ejemplo para las cadenas .ªbracadabraƶ .ªlabaralabarda"se necestia 7 operaciones.

- Jaccard : Es una medida de similitud que esta definido por defined el tamaño de la intersección de dos secuencias divido por el tamaño de la union de ambas secuencias, un ejemplo en la medida de similitud entre las secuencias "nightƶ "nacht.ᵉˢ de 0.3.

- Distancia Hamming : Se tiene dos cadenas de caracteres de igual tamaño , es la cantidad de sustituciones necesarios para transformar de una cadena de caracteres a otra. Por ejemplo "gatoƶ "pera"se necesita 4 sustituciones.

La medida de similitud implementado para el algoritmo de agrupacion es una distancia de compresion, utilizando algun metodo de compresion como lzma,gzip o bzip.se aplica la siguiente formula para obtener la *distancia* 3.1 :

$$distancia = \frac{(d12 - d2)}{d1} \tag{3.1}$$

donde :

d1 es el tamaño comprimido del documento mas grande

d2 es el tamaño comprimido del documento mas pequeño

d12 es el tamaño de la union de d1 y d2 comprimidos

La variable $d12$ depende de la similitud de los strings comprimidos, si los strings tienen un grado de similitud la compresion sera mucho mas efectiva ya que se necesita un diccionario mucho menor para comprimir, al contrario ocurre cuando los strings son muy distintos entre si.
Entre mas pequeño el valor significa que ambos documentos son muy similares, y entre mas grande los valores significa que los documentos son diferentes. En comparacion con la edicion de distancia tambien obtiene una buena calidad de similitud pero el tiempo de ejecucion es considerablemente mejor que edicion de distancia, lo que hace una buena opcion al momento de seleccionar una medida de similud para grandes colecciones de datos.

Por ejemplo, si se utiliza el compresor LZ78 en la secuencia $S1$ = *'abracadabra'* , $S2$ = *'abracadadah'* y la suma $S1S2$=*'abracadabraabracadadah'* se tiene: $S1$ = <0,a>,<0,b>,<0,r>,<1,c>,<1,d>,<1,b>,<3,a>de tamaño 7, $S2$ = <0,a>,<0,b>,<0,r>,<1,c>,<1,d>,<5,a>,<0,h>de tamaño 7, $S1S2$ = <0,a>,<0,b>,<0,r>,<1,c>,<1,d>,<1,b>,<3,a>,<6,r>,<4,a>,<0,d>,<5,a>,<0,h>de tamaño 12, al aplicar la formula 3.1 se obtiene el valor 0,71 .

Ahora que pasa si cambiamos la segunda secuencia a $S2$= 'casasyperro', que no tiene similitud con la primera secuencia : $S1$ = <0,a>,<0,b>,<0,r>,<1,c>,<1,d>,<1,b>,<3,
tamaño 7 $S2$ = <0,c>,<0,a><0,s>,<2,s>,<0,y>,<0,p>,<0,e>,<0,r>,<8,o>de tamaño 9 $S1S2$ = <0,a>,<0,b>,<0,r>,<1,c>,<1,d>,<1,b>,<3,a>,<0,c>,<1,s>,<9,y>,<9,p>
tamaño 14 se obtiene el valor 0.77, se observa que con menor similitud entre los strings mayor es el valor, entonces al tener strings que son similares el valor se aproxima al 0.

3.1.2. Algoritmo de agrupamiento propuesto

El algoritmo de agrupacion implementado para la distribucion de los string es una variante del algoritmo cure [**?**] que utiliza un algoritmo jerarquicos para formar los cluster iniciales.

A continuacion 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 de jerarquico.

2. Luego de formar los cluster iniciales se asigna cada string de la coleccion al cluster mas cercano.

3. Finalmente se comprime cada Cluster.

En el primer paso se debe seleccionar un algoritmo jerarquico , la implementacion es un algoritmo aglomerativo. Se ingresa cada string de la muestra a un cluster, y se calcula la similitud de los strings segun la medida de distancia nombrada anteriormente 3.1.1 , despues se selecciona ambos string que tengan la menor distancia y se juntan los clusters de cada string, antes de unirlos se pueden agregar varias reglas, por ejemplo, que las agrupaciones tengan un limite en el numero de strings, es medida puede servir para balancear los clusters.

En la segunda parte del algoritmo , a diferencia de cure que selecciona n puntos mas alejados entre si del clusters que representaran al cluster, llamados *puntos representativos*. El algoritmo toma todos los puntos como representativos. Unos de los objetivos que se busca es obtener un balance del espacio ocupado en cada agrupacion, para esto el documento antes de ser ingresado al grupo mas cercano se comprueba que:

$$cl < size(\frac{g}{n}) \tag{3.2}$$

donde :

cl Agrupacion mas cercano al documento

g espacio en memoria de la coleccion

n numero de agrupaciones

Para entender con mayor claridad el algoritmo propuesto se tiene el pseudo-codigo 1 .
El algoritmo 1 en la linea 1 es la muestra tomada de la coleccion de strings y en la linea 17 es la coleccion de strings. En el Algoritmo 2 muestra la funcion similitud.

Algorithm 1 Algoritmo de agrupamiento propuesto

1: $Sampling = \{d_1, \ldots, d_n\}$
2: $S \leftarrow \langle\ \rangle$
3: **for each** s1 in *Sampling* **do**
4:
5: **for each** s2 in *Sampling* **do**
6: ADD(S, [SIMILITUD(s1,s2) , s1 ,s2])
7: **end for**
8: **end for**
9: Sort(S)
10: $i = 0$
11: **while** Stop when j clusters in C **do**
12: $Cluster1 \leftarrow Sampling[S[i][1]]$
13: $Cluster2 \leftarrow Sampling[S[i][2]]$
14: MERGE(Cluster1,Cluster2)
15: $i = i + 1$
16: **end while**
17: $Collection = \{d_1, \ldots, d_k\}$
18: $Cluster \leftarrow sizej$
19: **for each** s in *Collection* **do**
20: $i \leftarrow ClusterconmayorSimilitudsenSampling$
21: ADD(Cluster[i],s)
22: **end for**
23: **for each** c in *Cluster* **do**
24: COMPRESS(c)
25: **end for**

Algorithm 2 Funcion SIMILITUD

Require: String1
Require: String2
1: $s1 \leftarrow COMPRESS(String1)$
2: $s2 \leftarrow COMPRESS(String2)$
3: $s1_2 \leftarrow COMPRESS(String1 + String2)$
4: **return** $size(s1_2) - size(s2)/size(s1)$

3.1.3. pruebas

Las pruebas se realizaran comparando dos tipos de algoritmo de agrupamiento :

- Algoritmo de agrupamiento propuesto
- Algoritmo de agrupamiento random

El Algoritmo de agrupamiento crea n clusters y asigna de manera aleatoria cada secuencia de string a un cluster a diferencia del algoritmo propuesto anteriormente que es determinista. Tambien manteniene el balance de espacio en memoria en cada cluster.
Con ambos algoritmo se pretende demostrar que agrupando los grupos de manera inteligente se pueda obtener mejores resultados que agrupandolos aleatoriamiente.
En el algoritmo de agrupamiento propuesto existen distintas variables que pueden determinar un buen agrupamiento de la coleccion tales como:

- Cantidad de clusters: Es dificil determinar la cantidad exacta de clusters que se necesita,
- 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 coleccion.
- Medida de similitud: Si la medida de similitud no obtiene una buana calidad es posible que agregue "puntos"de otro grupo, en la compresion se permite determinar el nivel de compresion, mietrnas que sea mayor el nivel la comresion es mayor pero es mucho mas lento.

Las pruebas se realizon de un dataset de Wikipedia que contiene 16384 documentos de 1 MB c/u, en su totalidad es de 16.384 GB . La coleccion esta formada por documentos versionados.

Capítulo 4

Resultados

En las pruebas se tomo una muesta no superior a 30 documentos ya que al aumetnar la muestra el tiempo de ejecucion crece exponencialmente. La muestra se obtuvo aleatoriamente de la coleccion de documentos, como la muestra es insignificante en comparacion al tamaño de la muestra es muy probable que ningun documentos perteneciera a un documento de la misma version. Este problema orgina que la mayoria de los documentos pertecescan a un solo grupo y el resto solamente es representado por 1 documento. Tambien cabe mencionar que la eleccion 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 numero 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 metodo con una cantidad de 10 agrupaciones, en cada agrupacion se muestra el resultado de la compresion. El $Metodo1$ utiliza el algoritmo de agrupacion aleatoria , en los metodos siguientes se utiliza el algoritmo de agrupacion propuesto pero modificando algunas variables para observar su comportamiento, mas adelante se menciona los cambios que se realizaron.

Clusters	Metodo 1(KiB)	Metodo 2(KiB)	Metodo 3(KiB)	Metodo 4(KiB)
Cluster 1	19.445	1.089	5.595	5.168
Cluster 2	19.445	1.442	7.102	8.271
Cluster 3	19.445	1.020	3.406	3.831
Cluster 4	19.445	2.851	3.465	3.353
Cluster 5	19.445	857	4.523	3.570
Cluster 6	19.445	1.455	2.729	4.343
Cluster 7	19.445	756	2.678	4.349
Cluster 8	19.445	1.106	2.748	3.239
Cluster 9	19.445	16.551	2.473	4.306
Cluster 10	19.445	2.245	1.318	5.209
Total	194.450	29.371	36.036	45.639

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

El $Metodo2$ se puede observar una gran mejora con respecto al algoritmo de agrupamiento aleatorio, aqui la muestra es de 30 documentos.
En el $Metodo3$ se balancea la cantidad de muestras en cada agrupacion. Para esto cada grupo no tendra una muestra de documentos superior a 3 en un universo de 30 documentos. Con esto se busca balancear la cantidad de documentos en cada agrupacion. En comparacion con el algoritmo de agrupamiento aleatorio sigue siendo una mejor alternativa pero comparando con los resulado del metodo 2 aumenta.
En el $Metodo4$, se hace la misma prueba que en el metodo anterior pero se agrega una condicion de que el tamaño en espacio de memoria no supere un limite, el limite en esta caso es el tamaño en memoria de la coleccion de datos divido por la cantidad de agrupaciones.

Capítulo 5

Conclusión

Se puede concluir que agrupando las secuencias de string de manera inteligente se obtiene mejores resultados que agrupandolos aleatoriamente, pero intentar balancear la carga de espacio en cada agrupacion se paga un costo al comprimir. Parte importante para obtener una buena agrupacion es la medida de similitud , para agrupar grandes cantidades de string es necesario que la medida de similitud entre dos string sea rapido 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 coleccion siempre entrega el mismo resulatdo, entonces al momento de obtener las agrupaciones con el sampling es posible asignar string en varios procesos rebajando el tiempo de ejecucion. El plan de trabajo para la segunda etapa de la tesis consistira en intentar mejorar los resultados de la compresion de la coleccion de datos obtenidos en esta primera parte , por ejemplo, cambiando las distintas variables que influyen en agrupacion de strings como el numero de agrupaciones , para obtener mejores resultados en la compresion de las agrupaciones.

Referencias bibliográficas

[1] D. Flanagan, *JavaScript*. O'Reilly Media, Inc., 1998.

[2] M. Stonebraker and G. Kemnitz, "The postgres next generation database management system," *Communications of the ACM*, vol. 34, no. 10, pp. 78–92, 1991.

[3] P. F. SCD, "Licencias, http://www.scd.cl/www/index.php/preguntas-frecuentes/," 2013.

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

PRIMER ANEXO

```
#!/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(compresse
        else:
                n = (len(compressed12) - len (compressed1)) / float(len(compress
        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()

        #Crear cluster
        #data=[["asdasdf"],["asdasdg"],["qweqweq"],["qweqwew"],["ghjghjk"],["ghj

        #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=editdist.distance(data[k][0],da
                        #comp=distance.hamming(data[k][0],dat
                        comp=distancia_zip(data[k][0],data[j]
                        print "t1:"+str(k)+" contra t2:"+str(
                        test.append([comp,data[k][0],data[j][
test.sort()

#print "los primeros 10 textos mas pequeos en distancia:"
#for h in range(20):
#       print test[h][0]
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(da
                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
                        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(
                        clust_size[s[1]]+=doc_z
                        #shutil.move(str(sys.argv[4])+str(docu
                        break
            f.close()

    for s in clust_size:
            print s

    comprimir()
```